MD5算法程序设计与实现

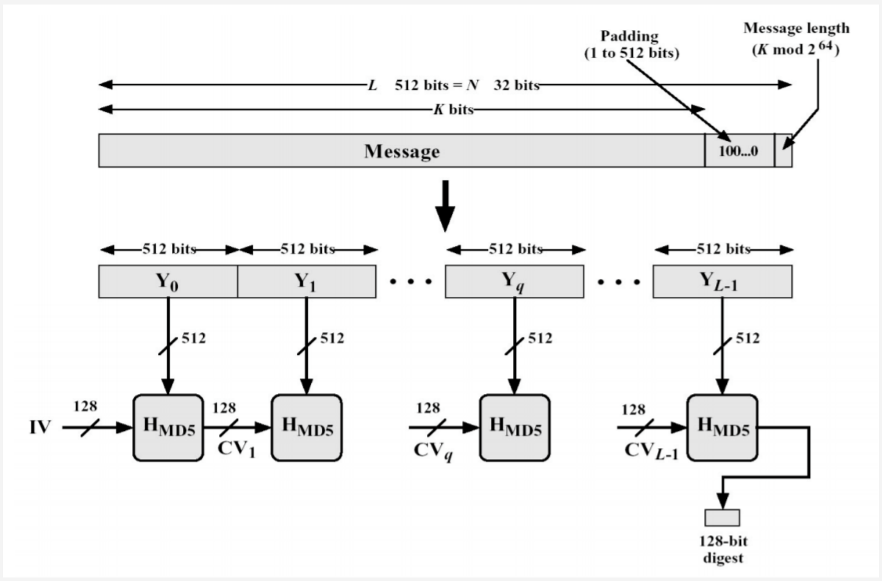
黄灿铭 16340079

算法原理概述

概述

- MD5 即 Message-Digest Algorithm 5 (信息-摘要算法 5)
- MD4 (1990)、MD5(1992, RFC 1321) 由 **Ron Rivest** 发明，是广泛使用的 **Hash 算法**，用于确保信息传输的完整性和一致性。
- MD5 使用 little-endian (小端模式)，输入任意不定长度信息，以512-bit 进行分组，生成四个32-bit 数据，最后联合输出固定128-bit 的信息摘要。
- MD5 算法的基本过程为：**填充、分块、缓冲区初始化、循环压缩、得出结果。**

基本流程



MD5算法逻辑

填充 padding

- 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).
    - 注意到当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 264)，最后得到一个长度位数为 K + P + 64 = 0 (mod 512) 的消息。
- 在C++中如何实现填充？（代码摘自MD5算法百度百科，由于代码实现很巧妙，自愧不如，遂主要对该代码进行学习，分析，吸收)

```
string str = "需要填充的原始消息数据"

unsigned int* padding(string str) {
 //分块:
 //由于char类型存储长度为8bits,
 //因此 TotalBits = str.length() * 8 为原始消息数据位数
 //假设 TotalBits / 512 = PreBlock 为未填充前的块数,
 //理论上填充后 block = PreBlock + 1
 //所以 block = (TotalBits + 64) / 512 + 1,
  //64为str的低64位，约分后得下式
  unsigned int block = ((str.length() + 8) / 64) + 1;

  //由于unsigned int类型存储长度为32位,
  //因此存放填充后的消息数据需要block * 512 / 32个unsigned int
  //即 strlength = block * 16
  strlength = block * 16;
  unsigned int *strByte = new unsigned int[strlength];

  for (unsigned int i = 0; i < strlength; i++)
  strByte[i] = 0; //初始化strByte数组

  //由于unsigned int类型存储长度为32位，因此能存储四个char类型
  //因此每四个str[i % 4]: ([i], [i + 1], [i + 2], [i + 3]),
  //存放在一个strByte[i >> 2]里
  //str[i]存放在strByte[i >> 2]的0-7位，[i+1]存放在8-15位,
  //[i+2]存放在16-23位，[i+3]存放在24-31位
  for (unsigned int i = 0; i < str.length(); i++) {
    strByte[i >> 2] |= (str[i]) << ((i % 4) * 8);
    }

    //在尾部添加1，因为str长度为字节数，转换为位数肯定为8的倍数，
    //所以可以直接在尾部添加0x80，其余位数由于初始化已为0，不需要再补0
    strByte[str.length() >> 2] |= 0x80 << (((str.length() % 4)) * 8);

    //添加原长度，长度指位的长度，所以要乘8，
    //由于是小端模式，所以放在倒数第二个,这里长度只用了32位
    strByte[strlength - 2] = str.length() * 8;
    return strByte;
}

```

分块

- 把填充后的消息结果分割为 L 个 512-bit 分组：Y0, Y1, ..., YL-1。
- 分组结果也可表示成 N 个32-bit 字 M0, M1, ..., MN-1， N = L*16。
- 如何实现?

```
//填充
unsigned int *strByte = padding(source);

//strlength存储单位为unsigned int,
//因此block_num = strlength * 32 / 512
for (unsigned int i = 0; i < strlength / 16; i++) {
 //16个unsigned int存储512位数
 unsigned int block[16];
 //将每一块数据存储到block里
  for (unsigned int j = 0; j < 16; j++)
  block[j] = strByte[i * 16 + j];
  //循环迭代每一块数据，每一次压缩算法后得到的128位数据
  //（a,b,c,d）将提供给下一次执行
  HMD5(block);
}
```

初始化

初始化一个128-bit 的 MD 缓冲区，记为 CVq，表示成4个32-bit寄存器 (A, B, C, D)；CV0 = 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。

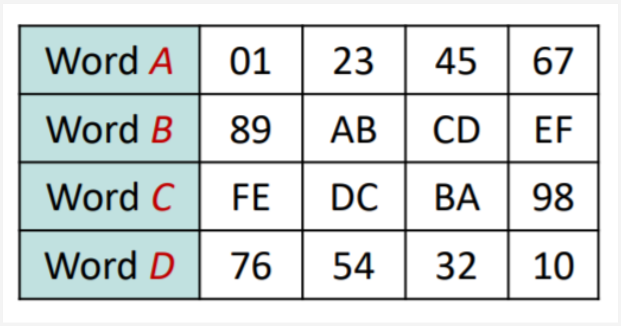
寄存器 (A, B, C, D) 置16进制初值作为初始向量 IV，并采用小端存储 (little-endian) 的存储结构：

A = 0x67452301

B = 0xEFCDAB89

C = 0x98BADCFE

D = 0x10325476



Word *A*	01	23	45	67
Word *B*	89	AB	CD	EF
Word *C*	FE	DC	BA	98
Word *D*	76	54	32	10

> Little-Endian 将低位字节排放在内存的低地址端，高位字节 排放在内存的高地址端。相反 Big-Endian 将高位字节排放在内存的低地址端，低位字节排放在内存的高地址端。存储结构与 CPU 体系结构和语言编译器有关。PowerPC 系列采用 Big Endian 方式存储数据，而 Intel x86系列则采用 Little Endian 方式存储。

```
#define A 0x67452301
#define B 0xefcdab89
#define C 0x98badcfe
#define D 0x10325476

atemp = A; //初始化
btemp = B;
ctemp = C;
dtemp = D;
 //a,b,c,d temp为每执行一次压缩算法后得到的128位数据
```

总控流程

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

CV0 = IV

CVi = HMD5(CVi-1 , Yi)

输出结果：

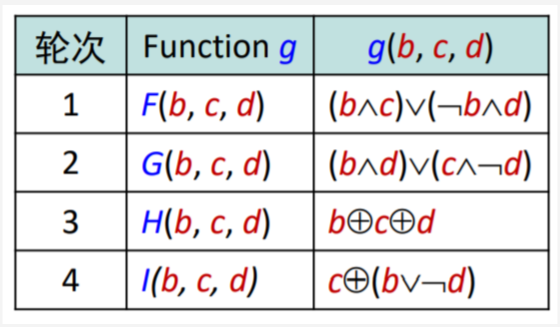
MD = CVL .

如何实现?

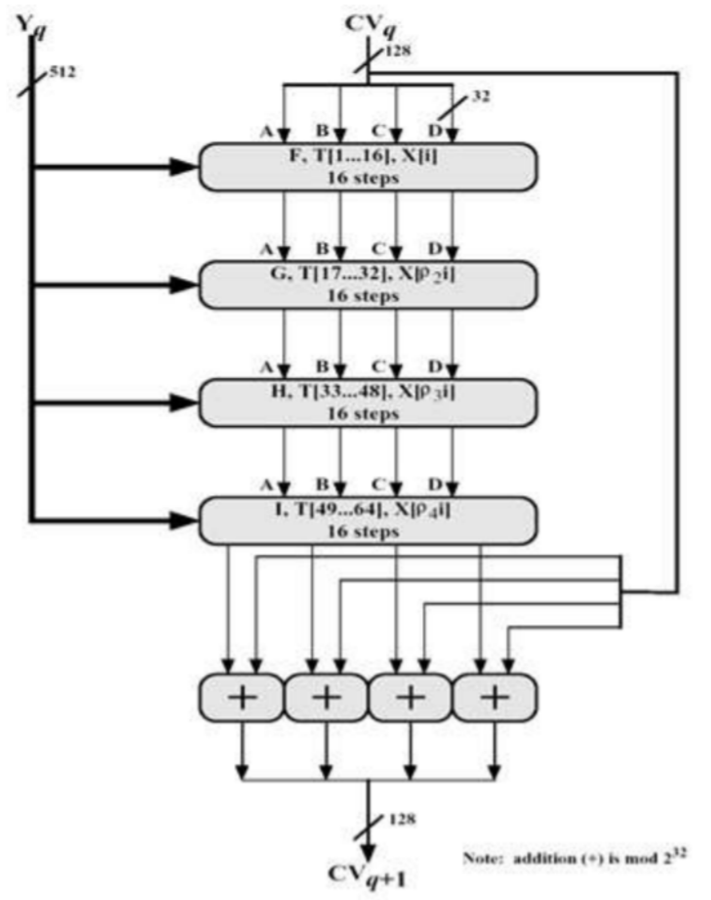
```
在代码中，L为(strlength / 16）

void HMD5(unsigned int X[]) {
 unsigned int a = atemp;
 unsigned int b = btemp;
 unsigned int c = ctemp;
 unsigned int d = dtemp;
 ...
 //对(a,b,c,d)和(X[]])循环压缩
  ...
  atemp = a + atemp;
  btemp = b + btemp;
  ctemp = c + ctemp;
  dtemp = d + dtemp;
  //a,b,c,d temp为每执行一次压缩算法后得到的128位数据
}
```

MD5 压缩函数 HMD5

- HMD5 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。
- 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。
- 4轮循环总共有64次迭代运算。4轮循环中使用的生成函数(轮函数) g 是一个32位非线性逻辑函数，在相应各轮的定义如下：



轮次	Function g	$g(b, c, d)$
1	$F(b, c, d)$	$(b \wedge c) \vee (\neg b \wedge d)$
2	$G(b, c, d)$	$(b \wedge d) \vee (c \wedge \neg d)$
3	$H(b, c, d)$	$b \oplus c \oplus d$
4	$I(b, c, d)$	$c \oplus (b \vee \neg d)$



MD5 第 *q* 分组的4轮循环逻辑 (压缩函数)

如何实现?

```
#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)))

for (unsigned int i = 0; i < 64; i++) {
 if (i < 16) {
 g = F(b, c, d);
 }
  else if (i < 32) {
  g = G(b, c, d);
  }
  else if (i < 48) {
  g = H(b, c, d);
  }
  else {
    g = I(b, c, d);
  }
}
```

每轮循环中的一次迭代运算逻辑

(1) 对 A 迭代：

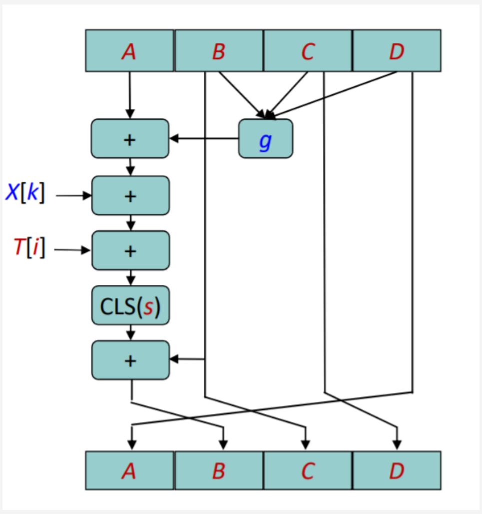
$$a = b + ((a + g(b, c, d) + X[k] + T[i]) <<<s)$$

(2) 缓冲区 (A, B, C, D) 作循环轮换：

$$(B, C, D, A) \longleftarrow (A, B, C, D)$$

> 说明：
> - a, b, c, d : MD 缓冲区 (A, B, C, D) 的当前值。
> - g : 轮函数 (F, G, H, I 中的一个)。
> - <<<s : 将32位输入循环左移 (CLS) s 位。
> - X[k] : 当前处理消息分组的第 k 个 (k =0...15) 32位字，即 Mq□16+k。
> - T[i] : T 表的第 i 个元素，32位字；T 表总共有64个元素，也 称为加法常数。
> - + : 模 2^32 加法。 每轮循环中的一次迭代运算逻辑

每轮循环中的一次迭代运算逻辑



如何实现？

```
#define CLS(x, n) (((x) << (n)) | ((x) >> (32-(n))))
//循环左移N位，即意味着X左移N位后，
//前N个被挤占的位数，移动到X的后N位，相当于X右移32-N位
//右移的时候，高位一定要补零，而不是补充符号位

unsigned int temp = d;
d = c;
```

```
c = b;
b = b + CLS((a + g + T[i] + X[k]), s[i]);
 a = temp;
```

各轮循环中第 i 次迭代 (i = 1...16) 使用的 X[k] 的确定

设 j = i -1 :

- 第1轮迭代：k = j.
  顺序使用 X[0, 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15]
- 第2轮迭代：k = (1 + 5j) mod 16.
  顺序使用 X[1, 6,11, 0, 5,10,15, 4, 9,14, 3, 8,13, 2, 7,12]
- 第3轮迭代：k = (5 + 3j) mod 16.
  顺序使用 X[5, 8,11,14, 1, 4, 7,10,13, 0, 3, 6, 9,12,15, 2]
- 第4轮迭代：k = 7j mod 16.
  顺序使用 X[0, 7,14, 5,12, 3,10, 1, 8,15, 6,13, 4,11, 2, 9]

如何实现?

```
for (unsigned int i = 0; i < 64; i++) {
 if (i < 16) {
 k = i;
 }
 else if (i < 32) {
 k = (5 * i + 1) % 16;
 }
 else if (i < 48) {
 k = (3 * i + 5) % 16;
  }
  else {
  k = (7 * i) % 16;
  }
}
```

T 表的生成

- T[i] = int(2^32 * |sin(i)|) • int 取整函数，sin 正弦函数，以 i 作为弧度输入。
- 各次迭代运算采用的 T 值：

```
const unsigned int T[] = {
 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 };
```

各次迭代运算采用的左循环移位的 s 值

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 }

总结：HMD5函数实现：

```
#define CLS(x, n) (((x) << (n)) | ((x) >> (32-(n))))
#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)))

void HMD5(unsigned int X[]) {
 unsigned int g, k;
 unsigned int a = atemp;
  unsigned int b = btemp;
  unsigned int c = ctemp;
  unsigned int d = dtemp;
  for (unsigned int i = 0; i < 64; i++) {
  if (i < 16) {
  g = F(b, c, d);
  k = i;
  }
  else if (i < 32) {
```

```
 g = G(b, c, d);
 k = (5 * i + 1) % 16;
 }
 else if (i < 48) {
 g = H(b, c, d);
 k = (3 * i + 5) % 16;
 }
 else {
 g = I(b, c, d);
 k = (7 * i) % 16;
 }
 unsigned int temp = d;
 d = c;
 c = b;
 b = b + CLS((a + g + T[i] + X[k]), s[i]);
 a = temp;
 }
 atemp = a + atemp;
 btemp = b + btemp;
 ctemp = c + ctemp;
 dtemp = d + dtemp;
}
```

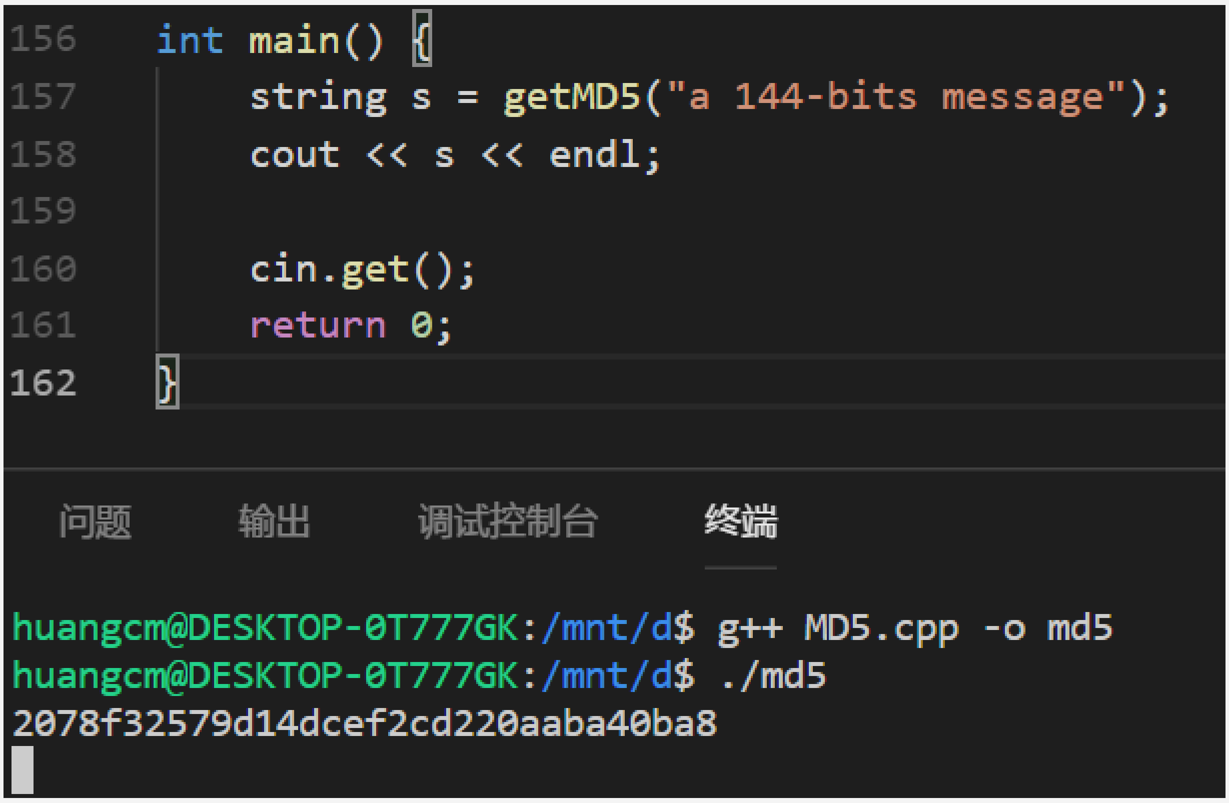
编译运行结果

TEST 1

1. 输入144 bits 的信息：

a 144-bits message

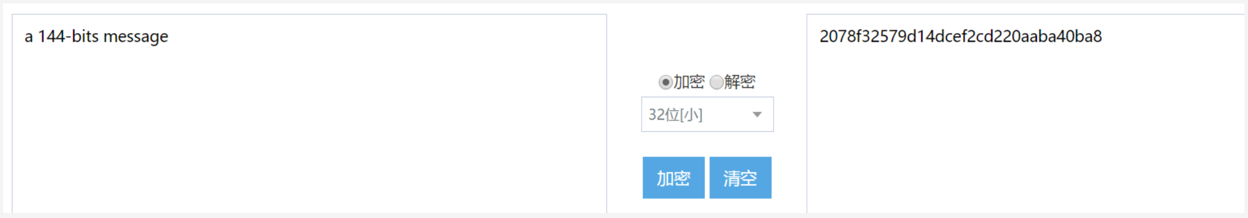
2. 运行结果:



```
int main() {
    string s = getMD5("a 144-bits message");
    cout << s << endl;

    cin.get();
    return 0;
}
```

```
huangcm@DESKTOP-0T777GK:/mnt/d$ g++ MD5.cpp -o md5
huangcm@DESKTOP-0T777GK:/mnt/d$ ./md5
2078f32579d14dcef2cd220aaba40ba8
```

3. 验证（32 位小写）：



a 144-bits message

[✓ 加密] 解密

32位[小]

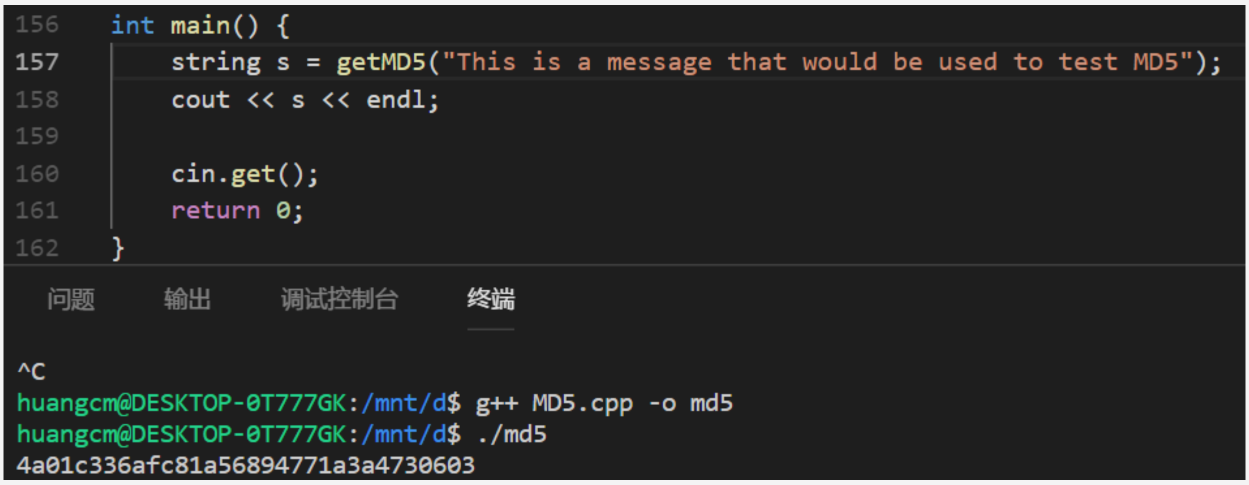
2078f32579d14dcef2cd220aaba40ba8

TEST 2

1. 输入384 bits 的信息:

This is a message that would be used to test MD5

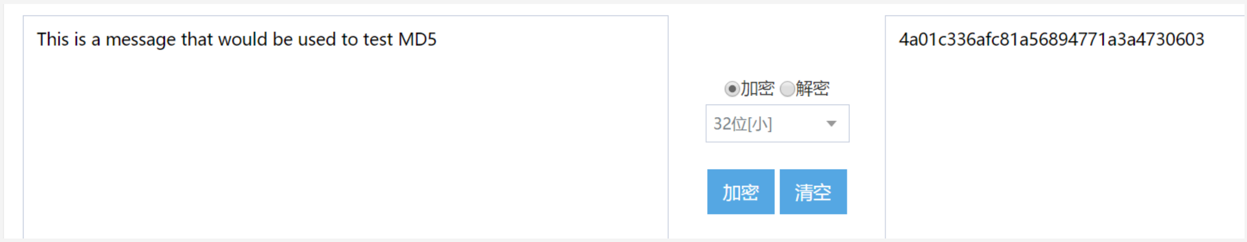
2. 运行结果:



```
int main() {
    string s = getMD5("This is a message that would be used to test MD5");
    cout << s << endl;

    cin.get();
    return 0;
}
```

```
^C
huangcm@DESKTOP-0T777GK:/mnt/d$ g++ MD5.cpp -o md5
huangcm@DESKTOP-0T777GK:/mnt/d$ ./md5
4a01c336afc81a56894771a3a4730603
```

3. 验证：



This is a message that would be used to test MD5

[✓ 加密] 解密

32位[小]

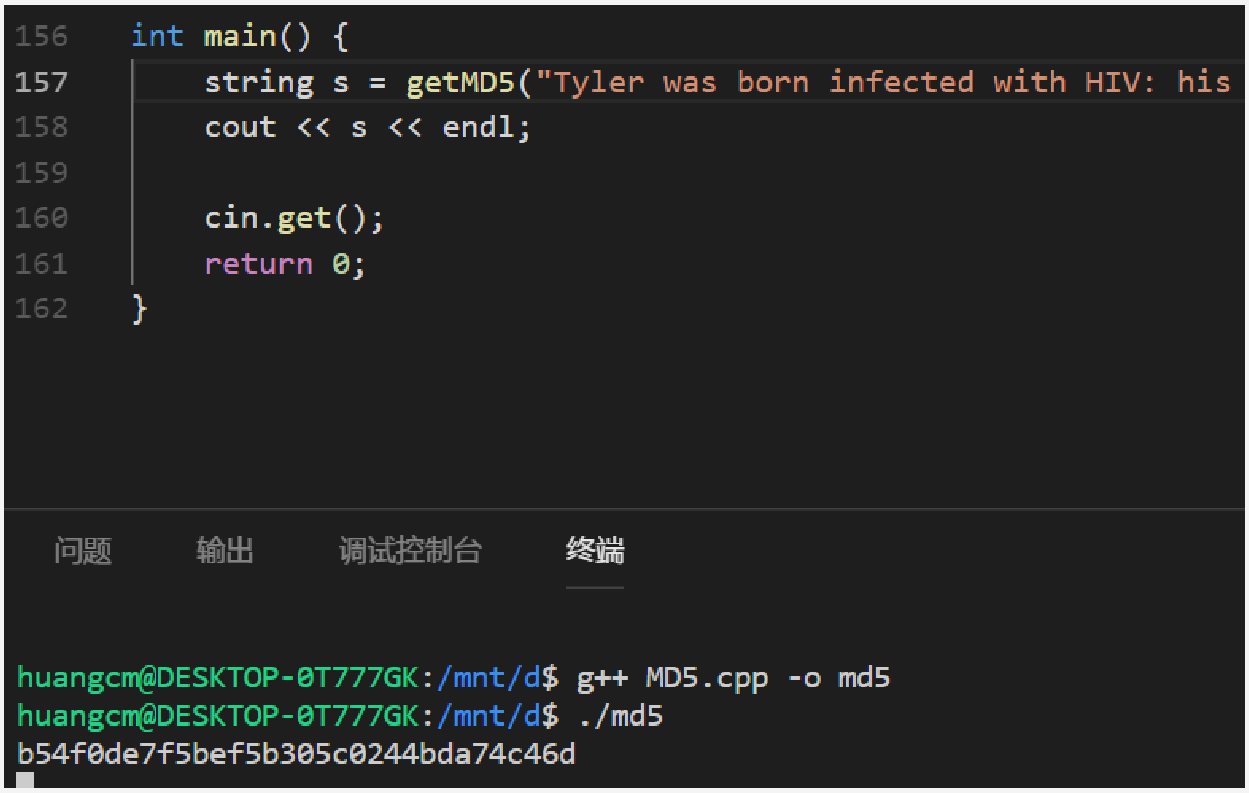
4a01c336afc81a56894771a3a4730603

TEST 3

1. 输入3992 bits 的信息：

Tyler was born infected with HIV: his mother was also infected. From the very beginning of his life, he was dependent on medications to enable him to survive. When he was five, he had a tube surgically inserted in a vein in his chest. This tube was connected to a pump, which he carried in a small backpack on his back. Medications were hooked up to this pump and were continuously supplied through this tube to his bloodstream. At times, he also needed supplemented oxygen to support his breathing.

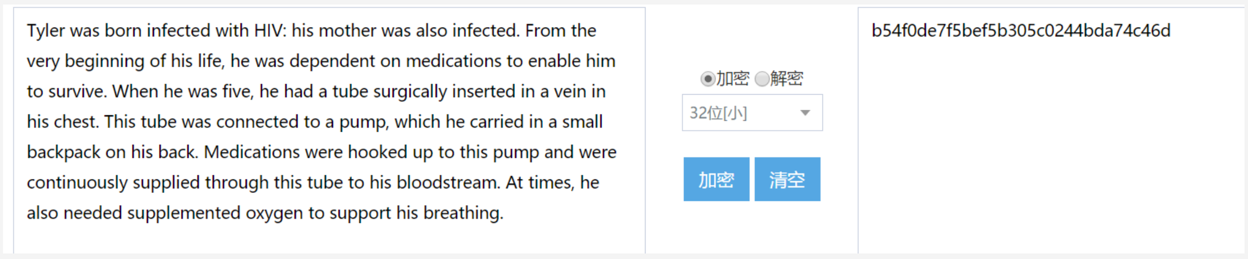
2. 运行结果：



```
int main() {
    string s = getMD5("Tyler was born infected with HIV: his
    cout << s << endl;

    cin.get();
    return 0;
}
```

```
huangcm@DESKTOP-0T777GK:/mnt/d$ g++ MD5.cpp -o md5
huangcm@DESKTOP-0T777GK:/mnt/d$ ./md5
b54f0de7f5bef5b305c0244bda74c46d
```

3.验证：



Tyler was born infected with HIV: his mother was also infected. From the very beginning of his life, he was dependent on medications to enable him to survive. When he was five, he had a tube surgically inserted in a vein in his chest. This tube was connected to a pump, which he carried in a small backpack on his back. Medications were hooked up to this pump and were continuously supplied through this tube to his bloodstream. At times, he also needed supplemented oxygen to support his breathing.

[✓ 加密] 解密

32位[小]

b54f0de7f5bef5b305c0244bda74c46d