南開大學
Nankai University

南　开　大　学

计 算 机 学 院

网络安全技术作业报告

基于 MD5 算法的文件完整性校验程序

朱浩泽 1911530

年级：2019 级

专业：计算机科学与技术

班级：计算机科学与技术 2 班

2022 年 5 月 20 日

目录

一、 实验目的

MD5 算法是目前最流行的一种信息摘要算法，在数字签名，加密与解密技术，以及文件完整性检测等领域中发挥着巨大的作用。熟悉 MD5 算法对开发网络应用程序，理解网络安全的概念具有十分重要的意义。

- 深入理解 MD5 算法的基本原理。
- 掌握利用 MD5 算法生成数据摘要的所有计算过程。
- 掌握 Linux 系统中检测文件完整性的基本方法。
- 熟悉 Linux 系统中文件的基本操作方法。

二、 实验目的

- 准确地实现 MD5 算法的完整计算过程。
- 对于任意长度的字符串能够生成 128 位 MD5 摘要。
- 对于任意大小的文件能够生成 128 位 MD5 摘要。
- 通过检查 MD5 摘要的正确性来检验原文件的完整性。

三、 实验步骤及实验结果

(一) 实验步骤

1. 实验环境

macOS 12.3(基于 unix), C++11, Cmake

2. 核心代码实现

- MD5 **类定义实现**

  在 MD5.h 头文件中实现 MD5 类，该类用于实现 MD5 算法的基本功能，包括 MD5 的计算过程，以及 MD5 的校验过程。

```
class MD5
{
private:
    DWORD state[4]; //用于表示 4 个初始向量
    DWORD count[2]; //用于计数，count[0]表示低位，count[1]表示高位
    BYTE buffer_block[64]; //用于保存计算过程中按块划分后剩下的比特流
    BYTE digest[16]; //用于保存 128 比特长度的摘要
    bool is_finished; //用于标志摘要计算过程是否结束
    static const BYTE padding[64]; //用于保存消息后面填充的数据块
    static const char hex[16]; //用于保存 16 进制的字符
    void Stop();
    void Transform(const BYTE block[64]);
    //将双字流转换为字节流
    void Encode(const DWORD *input, BYTE *output, size_t length);
    //将字节流转换为双字流
```

```
    void Decode(const BYTE *input, DWORD *output, size_t length);
    //将字节流按照十六进制字符串形式输出
    std::string BytesToHexString(const BYTE *input, size_t length);

public:
    MD5();
    MD5(const std::string &str);
    MD5(std::ifstream &in);
    void Update(std::ifstream &in);
    void Update(const BYTE* input,size_t length);
    const BYTE* GetDigest();
    std::string Tostring();
    void Reset();
};
```

对于私有变量

- 数组 state 表示四个初始向量
- 数组 count 是计数器，记录已经运算的比特数
- buffer_block 是 64 字节的缓存快，保存消息被划分后不足 64 字节的数据，或者保存每次运算的 64 字节数据
- digest 用于保存生成的 MD5 摘要
- padding 存储填充块
- hex 存储 16 个 16 进制字符

对于成员函数

- Update(const BYTE* input, size_t length) 对给定字符串进行 MD5 运算
- Update 对给定长度的字节流进行 MD5 运算
- get_digest 获取摘要
- Reset 重置初始变量
- Transform 对一个 512 比特的消息分组进行 MD5 运算
- Decode 将 64byte 的数据块划分为 16 个 32bit 大小的子分组
- Tostring 生成 MD5 摘要字符串

- **宏定义**

用宏定义定义四轮计算中的 FF、GG、HH、II 函数，其中 a、b、c、d 表示计算向量，x 表示一个 32 位的子块，s 表示循环左移的位数，ac 表示弧度。于此同时，定义在 MD5 四轮迭代计算中向量 A、B、C、D、循环左移的位数。

```
typedef unsigned char BYTE;
typedef unsigned long DWORD;

#define BUFFER_SIZE 8

#define S11 7
#define S12 12
#define S13 17
#define S14 22
```

```
#define S21 5
#define S22 9
#define S23 14
#define S24 20
#define S31 4
#define S32 11
#define S33 16
#define S34 23
#define S41 6
#define S42 10
#define S43 15
#define S44 21

#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 ROTATE_LEFT(x, n) (((x) << (n)) | ((x) >> (32-(n))))
#define FF(a, b, c, d, x, s, ac) {(a) += F ((b), (c), (d)) + (x) + ac;(a) =
    ROTATE_LEFT ((a), (s));(a) += (b);}
#define GG(a, b, c, d, x, s, ac) {(a) += G ((b), (c), (d)) + (x) + ac;(a) =
    ROTATE_LEFT ((a), (s));(a) += (b);}
#define HH(a, b, c, d, x, s, ac) {(a) += H ((b), (c), (d)) + (x) + ac;(a) =
    ROTATE_LEFT ((a), (s));(a) += (b);}
#define II(a, b, c, d, x, s, ac) {(a) += I ((b), (c), (d)) + (x) + ac;(a) =
    ROTATE_LEFT ((a), (s));(a) += (b);}
```

- Update **函数**

  将输入 MD5 哈希的字符串进行分块，把输入按 64 字节分为 N+1 个组，最后一个组不足 64 字节 (可能为 0 字节)，利用 count 数组和 buffer_block 数组记录剩余未加密的部分，然后每次加密时核对遗留未加密的数据块，填充后满足一个分组后，传入到 Transform 内核加密函数，得到的结果被存在 state 状态数组中，满足下次加密的时候自动在已经加密的部分继续操作。

```
void MD5::Update(const BYTE* input,size_t length)
{
   DWORD i, index, partLen;
   is_finished = false;
   index = (DWORD)((count[0] >> 3) & 0x3f);
   if((count[0] += ((DWORD)length << 3)) < ((DWORD)length << 3)) {
      count[1]++;
   }
   count[1] += ((DWORD)length >> 29);
   partLen = 64 - index;
   if(length >= partLen){
      memcpy(&buffer_block[index], input, partLen);
      Transform(buffer_block);
      for (i = partLen; i + 63 < length; i += 64) {
         Transform(&input[i]);
      }
      index = 0;
   }
   else {
```

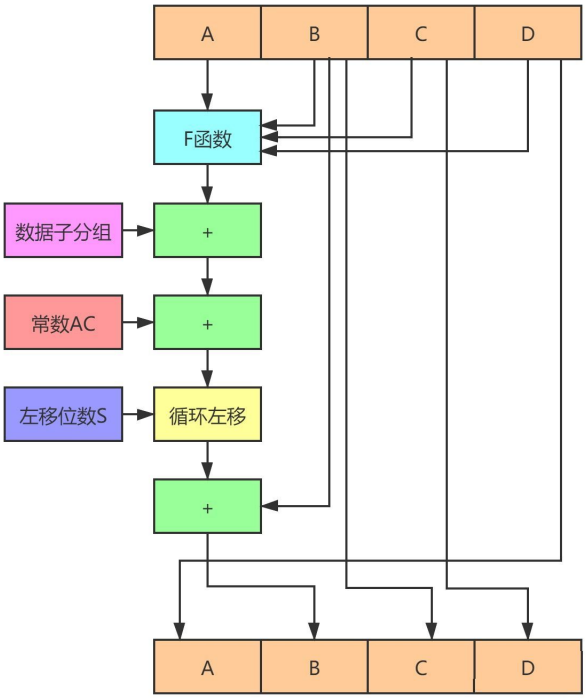
```
        i = 0;
    }
    memcpy(&buffer_block[index], &input[i], length-i);
}
```

我们也可以对 Update 函数进行重载，让其可以直接直接对文件流进行处理。

```
void MD5::Update(std::ifstream &in) {
    if (!in)
        return;
    std::streamsize length;
    char buffer[BUFFER_SIZE];
    while (!in.eof()) {
        in.read(buffer, BUFFER_SIZE);
        length = in.gcount();
        if (length > 0)
            Update((const BYTE*)buffer, length);
    }
    in.close();
}
```

- Transform **函数**

  循环变换是整个 MD5 算法最核心，也是最复杂的部分。一个 512 位分组的数据被进一步划分为 16 个 32 位的子分组，对每个子分组进行下图所示的变换：



图中的 F 函数代表一次由位运算构成的非线性变换，每一轮循环变换用到的 F 函数不一样。加号表示加法运算。常数 AC 的值在每一次变换中都不一样，表达式为 $AC_i = int(4294967296|sin(i)|)$ ，i 表示第 i 次变换。左移位数 S 有规律地周期性变化。数据的 16 个子分组都参与到上图所示的变换，顺序不定。当 16 个子分组处理完成时，我们就说完成

了一轮循环变换。MD5 的一个数据分组一共需要进行四轮的循环变换。将四轮循环变换后得到的 A、B、C、D 的值分别和原来的值相加，就是 A、B、C、D 进行循环变换后的结果。

该函数函数中进行了一系列 MD5 算法的内核运算。初始设置一个初始状态，然后对于这四个向量的初始向量进行四轮函数混淆加密，然后每一轮内部又进行很多函数操作。首先将初始向量 state 的数值赋给变量 a、b、c、d 中。调用 Decode 函数，将 64 字节的数据块划分为 16 个 32 比特大小的子分组。因为每一轮计算都是对 32 比特子分组进行操作，所以重新划分后可以方便后面的计算过程。依次调用函数 FF、GG、HH、II 展开 4 轮计算，其中每一轮计算包含 16 小步，每一步对一个 32 比特子分组进行运算。函数 FF、GG、HH、II 的前 4 个参数是变量 a、b、c、d 的不同排列，参数 X[k] 表示对第 k 个子分组进行计算，Sij 表示第 i 轮第 j 步计算循环左移的位数。

```
void MD5::Transform(const BYTE block[64]) {
    DWORD a = state[0], b = state[1], c = state[2], d = state[3], x[16];
    Decode(block, x, 64);
    /* 第 1 轮 */
    FF (a, b, c, d, x[ 0], S11, 0xd76aa478); /* 1 */
    FF (d, a, b, c, x[ 1], S12, 0xe8c7b756); /* 2 */
    FF (c, d, a, b, x[ 2], S13, 0x242070db); /* 3 */
    FF (b, c, d, a, x[ 3], S14, 0xc1bdceee); /* 4 */
    FF (a, b, c, d, x[ 4], S11, 0xf57c0faf); /* 5 */
    FF (d, a, b, c, x[ 5], S12, 0x4787c62a); /* 6 */
    FF (c, d, a, b, x[ 6], S13, 0xa8304613); /* 7 */
    FF (b, c, d, a, x[ 7], S14, 0xfd469501); /* 8 */
    FF (a, b, c, d, x[ 8], S11, 0x698098d8); /* 9 */
    FF (d, a, b, c, x[ 9], S12, 0x8b44f7af); /* 10 */
    FF (c, d, a, b, x[10], S13, 0xffff5bb1); /* 11 */
    FF (b, c, d, a, x[11], S14, 0x895cd7be); /* 12 */
    FF (a, b, c, d, x[12], S11, 0x6b901122); /* 13 */
    FF (d, a, b, c, x[13], S12, 0xfd987193); /* 14 */
    FF (c, d, a, b, x[14], S13, 0xa679438e); /* 15 */
    FF (b, c, d, a, x[15], S14, 0x49b40821); /* 16 */

    /* 第 2 轮 */
    GG (a, b, c, d, x[ 1], S21, 0xf61e2562); /* 17 */
    GG (d, a, b, c, x[ 6], S22, 0xc040b340); /* 18 */
    GG (c, d, a, b, x[11], S23, 0x265e5a51); /* 19 */
    GG (b, c, d, a, x[ 0], S24, 0xe9b6c7aa); /* 20 */
    GG (a, b, c, d, x[ 5], S21, 0xd62f105d); /* 21 */
    GG (d, a, b, c, x[10], S22, 0x2441453); /* 22 */
    GG (c, d, a, b, x[15], S23, 0xd8a1e681); /* 23 */
    GG (b, c, d, a, x[ 4], S24, 0xe7d3fbc8); /* 24 */
    GG (a, b, c, d, x[ 9], S21, 0x21e1cde6); /* 25 */
    GG (d, a, b, c, x[14], S22, 0xc33707d6); /* 26 */
    GG (c, d, a, b, x[ 3], S23, 0xf4d50d87); /* 27 */
    GG (b, c, d, a, x[ 8], S24, 0x455a14ed); /* 28 */
    GG (a, b, c, d, x[13], S21, 0xa9e3e905); /* 29 */
    GG (d, a, b, c, x[ 2], S22, 0xfcefa3f8); /* 30 */
    GG (c, d, a, b, x[ 7], S23, 0x676f02d9); /* 31 */
    GG (b, c, d, a, x[12], S24, 0x8d2a4c8a); /* 32 */

    /* 第 3 轮 */
    HH (a, b, c, d, x[ 5], S31, 0xfffa3942); /* 33 */
    HH (d, a, b, c, x[ 8], S32, 0x8771f681); /* 34 */
```

```
    HH (c, d, a, b, x[11], S33, 0x6d9d6122); /* 35 */
    HH (b, c, d, a, x[14], S34, 0xfde5380c); /* 36 */
    HH (a, b, c, d, x[ 1], S31, 0xa4beea44); /* 37 */
    HH (d, a, b, c, x[ 4], S32, 0x4bdecfa9); /* 38 */
    HH (c, d, a, b, x[ 7], S33, 0xf6bb4b60); /* 39 */
    HH (b, c, d, a, x[10], S34, 0xbebfbc70); /* 40 */
    HH (a, b, c, d, x[13], S31, 0x289b7ec6); /* 41 */
    HH (d, a, b, c, x[ 0], S32, 0xeaa127fa); /* 42 */
    HH (c, d, a, b, x[ 3], S33, 0xd4ef3085); /* 43 */
    HH (b, c, d, a, x[ 6], S34, 0x4881d05); /* 44 */
    HH (a, b, c, d, x[ 9], S31, 0xd9d4d039); /* 45 */
    HH (d, a, b, c, x[12], S32, 0xe6db99e5); /* 46 */
    HH (c, d, a, b, x[15], S33, 0x1fa27cf8); /* 47 */

    /* 第 4 轮 */
    II (a, b, c, d, x[ 0], S41, 0xf4292244); /* 49 */
    II (d, a, b, c, x[ 7], S42, 0x432aff97); /* 50 */
    II (c, d, a, b, x[14], S43, 0xab9423a7); /* 51 */
    II (b, c, d, a, x[ 5], S44, 0xfc93a039); /* 52 */
    II (a, b, c, d, x[12], S41, 0x655b59c3); /* 53 */
    II (d, a, b, c, x[ 3], S42, 0x8f0ccc92); /* 54 */
    II (c, d, a, b, x[10], S43, 0xffeff47d); /* 55 */
    II (b, c, d, a, x[ 1], S44, 0x85845dd1); /* 56 */
    II (a, b, c, d, x[ 8], S41, 0x6fa87e4f); /* 57 */
    II (d, a, b, c, x[15], S42, 0xfe2ce6e0); /* 58 */
    II (c, d, a, b, x[ 6], S43, 0xa3014314); /* 59 */
    II (b, c, d, a, x[13], S44, 0x4e0811a1); /* 60 */
    II (a, b, c, d, x[ 4], S41, 0xf7537e82); /* 61 */
    II (d, a, b, c, x[11], S42, 0xbd3af235); /* 62 */
    II (c, d, a, b, x[ 2], S43, 0x2ad7d2bb); /* 63 */
    II (b, c, d, a, x[ 9], S44, 0xeb86d391); /* 64 */
    state[0] += a;
    state[1] += b;
    state[2] += c;
    state[3] += d;
}
```

- Reset **函数**

  初始化函数，将记忆元件置零，初始化向量进行置位。

```
void MD5::Reset() {
    is_finished = false;
    count[0] = count[1] = 0;
    state[0] = 0x67452301;
    state[1] = 0xefcdab89;
    state[2] = 0x98badcfe;
    state[3] = 0x10325476;
}
```

- Stop **函数**

  用于终止摘要计算过程，输出摘要。具体来说是进行补充尾部以及进行最后的运算。首先是计算信息总长度和最后一个分组长度，进而计算出需要补足的长度。第一次调用 Update 函数是补充需要补充的长度并进行一轮运算。第二次调用 Update 函数是补充信息长度到末尾并进行最后一次运算。

```
void MD5::Stop() {
   BYTE bits[8];
   DWORD oldState[4];
   DWORD oldCount[2];
   DWORD index, padLen;
   memcpy(oldState, state, 16);
   memcpy(oldCount, count, 8);
   Encode(count, bits, 8);
   index = (DWORD)((count[0] >> 3) & 0x3f);
   padLen = (index < 56) ? (56 - index) : (120 - index);
   Update(padding, padLen);
   Update(bits, 8);
   Encode(state, digest, 16);
   memcpy(state, oldState, 16);
   memcpy(count, oldCount, 8);
}
```

- Tostring **函数**

  将摘要转换成字符串

```
std::string MD5::Tostring() {
   return BytesToHexString(GetDigest(), 16);
}

std::string MD5::BytesToHexString(const BYTE *input, size_t length) {
   std::string str;
   str.reserve(length << 1);
   for(size_t i = 0; i < length; i++) {
      int t = input[i];
      int a = t / 16;
      int b = t % 16;
      str.append(1, hex[a]);
      str.append(1, hex[b]);
   }
   return str;
}
```

- Decode **函数**

  将 64 字节 (64*8 bit) 的数据块划分为 16 个 32bit 大小的子分组

```
void MD5::Decode(const BYTE *input, DWORD *output, size_t length) {
   for(size_t i=0, j=0; j<length; i++, j+=4) {
      output[i] = ((DWORD)input[j]) | (((DWORD)input[j+1]) << 8) |
         (((DWORD)input[j+2]) << 16) | (((DWORD)input[j+3]) << 24);
   }
}
```

- Encode **函数**

  得到最终的结果

```
void MD5::Encode(const DWORD *input, BYTE *output, size_t length) {
   for(size_t i=0, j=0; j<length; i++, j+=4) {
      output[j]= (BYTE)(input[i] & 0xff);
      output[j+1] = (BYTE)((input[i] >> 8) & 0xff);
```

```
        output[j+2] = (BYTE)((input[i] >> 16) & 0xff);
        output[j+3] = (BYTE)((input[i] >> 24) & 0xff);
    }
}
```

3. 控制台函数实现

设计的 MD5 工具包括以下功能: 计算字符串的 MD5 值、计算文件的 MD5 值、文件的 MD5 值校验。

通过命令行参数的形式使用该工具，命令可选参数主要包括:-h、-t、-c、-v、-f、-s。

- -h 是用来显示帮助信息，显示出工具的所有参数以及其基本的使用格式。只需要简单地进行打印即可，我们在 main.cpp 文件中利用 print_h 函数实现

```
void print_h(int argc, char *argv[]) {
    if (2 != argc) {
        std::cout << "参数错误." << std::endl;
        return;
    }
    std::cout << "MD5: usage:\n" << "\t" << "[-h] --help information " << std::endl
        ;
    std::cout << "\t" << "[-t] --test MD5 application" << std::endl;
    std::cout << "\t" << "[-c] [file path of the file computed]" << std::endl;
    std::cout << "\t" << "\t" <<"--compute MD5 of the given file" << std::endl;
    std::cout << "\t" << "[-v] [file path of the file validated]" << std::endl;
    std::cout << "\t" << "\t" <<"--validate the integrality of a given file by
        manual input MD5 value" << std::endl;
    std::cout << "\t" << "[-f] [file path of the file validated] [file path of the
         .md5 file]" << std::endl;
    std::cout << "\t" << "\t" <<"--validate the integrality of a given file by
        read MD5 value from .md5 file" << std::endl;
}
```

]

- -t 是用来测试 MD5 工具的功能，可以通过该参数来测试 MD5 工具的功能。我们在 main.cpp 文件中利用 print_t 函数实现

```
void print_t(int argc, char *argv[]) {
    if (2 != argc) {
        std::cout << "参数错误." << std::endl;
        return;
    }
    std::string test[] = {"", "a", "abc", "message digest", "
        abcdefghijklmnopqrstuvwxyz",
    "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789",
    "
        12345678901234567890123456789012345678901234567890123456789012345678901234567890
        "
    };
    MD5 md5;
    for (int i = 0; i < 7; ++i) {
        md5.Update((const BYTE*)test[i].c_str(), test[i].length());
        std::cout << "MD5(\"" + test[i] + "\") = " << md5.Tostring()<< std::endl;
```

```
    }
}
```

- -c 是用来计算文件摘要并打印的。在 -c 参数后面写入文件的路径，首先读入文件，然后利用 MD5 类进行 MD5 值的计算，即可获得此文件的 MD5 值，计算的结果将会显示在终端屏幕上。我们在 main.cpp 文件中利用 print_c 函数实现

```
void print_c(int argc, char *argv[]) {
    if (3 != argc) {
        std::cout << "参数错误." << std::endl;
        return;
    }
    std::string filePath = argv[2];
    std::ifstream fileStream(filePath);
    MD5 md5;
    md5.Update(fileStream);
    std::cout << "The MD5 value of file(\"" << filePath << "\") is " << md5.
        Tostring() << std::endl;
}
```

- -v 参数用于比对文件的 MD5 哈希值。首先手动读入待比对的 MD5 值，并打印在终端上。然后读取待比对文件，利用 MD5 类进行 MD5 值的计算，将两个 MD5 进行比较后，将对比的结果输出在终端上。通过手动输入待比对的 MD5 哈希值，从而与文件计算出的 MD5 哈希值进行比较，得出文件是否出现更改，实现文件完整性校验。

  首先比较参数 argv[1]，判断是否通过手工输入进行验证。若是，则继续下面步骤; 否则，退出。检测被测文件的路径是否存在，若存在，则继续下面步骤; 否则，退出。输入被测文件的 MD5 摘要并保存在数组 InputMD5 中。打开被测文件，读取被测文件内容，并调用 Update 函数重新计算被测文件的 MD5 摘要。调用 Tostring 函数将 MD5 摘要表示成 16 进制字符串形式。最后调用 strcmp 函数判断两个摘要是否相同，若相同，则说明被测文件是完整的；否则，说明文件受到了破坏。

```
void print_v(int argc,char *argv[]) {
    if (3 != argc) {
        std::cout << "参数错误." << std::endl;
        return;
    }
    std::string filePath = argv[2];
    std::cout << "Please input the MD5 value of file(\"" << filePath << "\")..."
        << std::endl;
    std::string inputMD5;
    std::cin >> inputMD5;
    std::cout << "The old MD5 value of file(\"" << filePath << "\") you have input
         is" << std::endl << inputMD5 << std::endl;
    std::ifstream fileStream(filePath);
    MD5 md5;
    md5.Update(fileStream);
    std::string genMD5 = md5.Tostring();
    std::cout << "The new MD5 value of file(\"" << filePath << "\") that has
        computed is" << std::endl << genMD5 << std::endl;
    if (!genMD5.compare(inputMD5)) {
        std::cout << "OK! The file is integrated" << std::endl;
    }
```

```
    else {
        std::cout << "Match Error! The file has been modified!" << std::endl;
    }
}
```

- -f 输入文件和 MD5 文件，前者计算出摘要与后者比较。程序读取.md5 摘要，重新计算被测文件的 MD5，最后将两者逐位比较。首先比较参数 argv[1]，判断是否通过.md5 文件进行验证。若是，则继续下面步骤; 否则，退出。检测被测文件的路径和.md5 文件的路径是否存在，若存在，则继续下面步骤; 否则，退出。打开.md5 文件，读取文件中的记录，调用 strtok 函数获得被测文件的 MD5 摘要。打开被测文件，读取被测文件内容，并调用 Update 函数重新计算被测文件的 MD5 摘要。调用 Tostring 函数将 MD5 摘要表示成 16 进制字符串形式。最后调用 strcmp 函数判断两个摘要是否相同，若相同，则说明被测文件是完整的; 否则，说明文件受到了破坏。

```
void print_f(int argc, char *argv[]) {
    if (4 != argc) {
        std::cout << "参数错误." << std::endl;
        return;
    }
    std::string filePath = argv[2];
    std::string md5Path = argv[3];
    std::ifstream md5Stream(md5Path);
    std::string oldMD5Str((std::istreambuf_iterator<char>(md5Stream)), std::
        istreambuf_iterator<char>());
    oldMD5Str = (std::string)strtok(const_cast<char*>(oldMD5Str.c_str())," ");
    std::cout << "The old MD5 value of file(\"" << filePath << "\") in " <<
        md5Path << " is " << std::endl << oldMD5Str << std::endl;
    std::ifstream fileStream(filePath);
    MD5 md5;
    md5.Update(fileStream);
    std::string genMD5 = md5.Tostring();
    std::cout << "The new MD5 value of file(\"" << filePath << "\") that has
        computed is" << std::endl << genMD5 << std::endl;
    if (!genMD5.compare(oldMD5Str)) {
        std::cout << "OK! The file is integrated" << std::endl;
    }
    else {
        std::cout << "Match Error! The file has been modified!" << std::endl;
    }
}
```

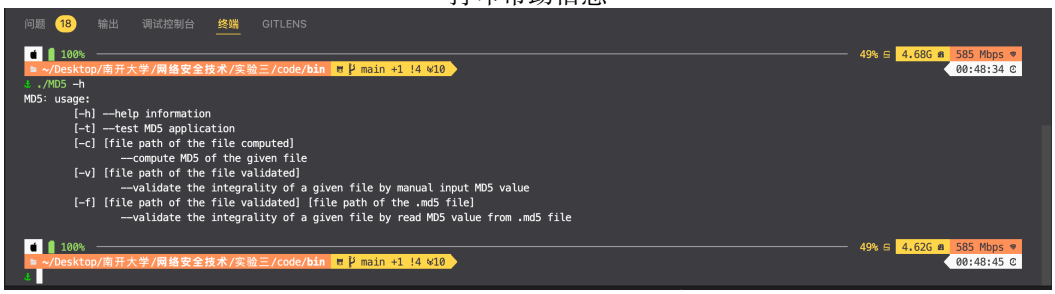
- 主函数

```
int main(int argc,char *argv[]) {
    std::unordered_map<std::string, void(*)(int, char*[])> mapOp = {{"-t", print_t
        }, {"-h", print_h}, {"-c", print_c}, {"-v", print_v}, {"-f", print_f}};
    if (argc < 2) {
        std::cout << "参数错误, argc = " << argc << std::endl;
        return -1;
    }
    std::string op = argv[1];
    if (mapOp.find(op) != mapOp.end()) {
        mapOp[op](argc, argv);
    }
```

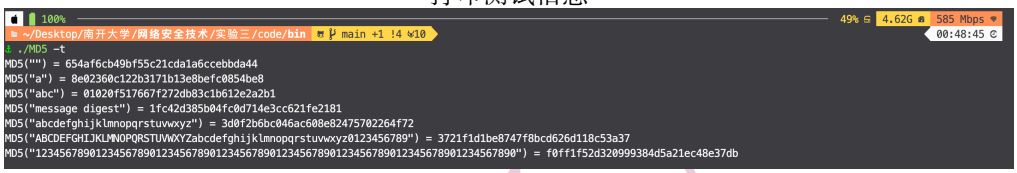
```
11      return 0;
12  }
```

(二) 实验结果

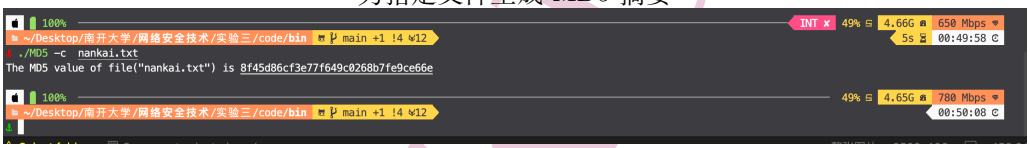
打印帮助信息



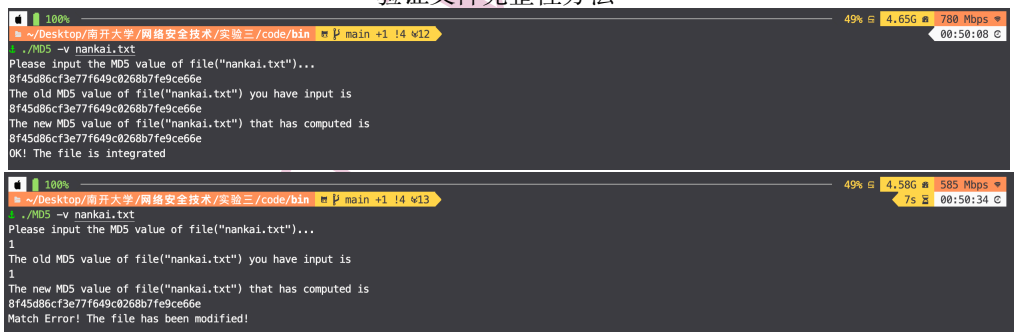
打印测试信息



为指定文件生成 MD5 摘要



验证文件完整性方法一



验证文件完整性方法二

四、 实验遇到的问题及解决方法

本次实验指导书对算法描述的比较详细，代码也给出了一定的框架，所以实现起来并没有什么结构性的大的问题，只有一些小的细节没有注意到。

1. 我们在使用 unordered_map 数据结构存储 string 类型和函数类型的变量时，出现了以下报错

```
1 warning generated.
[ 66%] Building CXX object CMakeFiles/main.dir/src/MD5.cpp.o
[100%] Linking CXX executable bin/main
Undefined symbols for architecture arm64:
  "MD5::Update(std::__1::basic_ifstream<char, std::__1::char_traits<char> >&)", referenced from:
      _main in main.cpp.o
ld: symbol(s) not found for architecture arm64
clang: error: linker command failed with exit code 1 (use -v to see invocation)
make[2]: *** [bin/main] Error 1
make[1]: *** [CMakeFiles/main.dir/all] Error 2
make: *** [all] Error 2
```

经过查阅资料，我们发现，该类型的 stl 容器要想匹配该功能，需要 C++11 以及以上的版本才可以实现，所以我们在 CmakeLists 中添加 `set (CMAKE_CXX_STANDARD 11)`，指定 C++ 版本为 C++11 后，重新生成 makefile，便解决了问题。

2. 我们在 MD5 算法类完成的时候，进行了测试，发现了如下报错

```
问题 22  输出  调试控制台  终端  GITLENS

[ 66%] Building CXX object CMakeFiles/MD5.dir/src/MD5.cpp.o
[100%] Linking CXX executable bin/MD5
Undefined symbols for architecture arm64:
  "MD5::hex", referenced from:
      MD5::BytesToHexString(unsigned char const*, unsigned long) in MD5.cpp.o
  "MD5::padding", referenced from:
      MD5::Stop() in MD5.cpp.o
ld: symbol(s) not found for architecture arm64
clang: error: linker command failed with exit code 1 (use -v to see invocation)
make[2]: *** [bin/MD5] Error 1
make[1]: *** [CMakeFiles/MD5.dir/all] Error 2
make: *** [all] Error 2
```

根据报错内容，我们可以看到，报错信息是我们没有在 MD5 类中声明 hex 和 padding。但是我们仔细检查了 MD5 类的源代码，类中确实定义了 hex 和 padding 变量。

```
private:
    DWORD state[4];  //用于表示 4 个初始向量
    DWORD count[2];  //用于计数，count[0]表示低位，count[1]表示高位
    BYTE buffer_block[64];  //用于保存计算过程中按块划分后剩下的比特流
    BYTE digest[16];  //用于保存 128 比特长度的摘要
    bool is_finished;  //用于标志摘要计算过程是否结束
    static const BYTE padding[64];  //用于保存消息后面填充的数据块
    static const char hex[16]; //用于保存 16 进制的字符
```

经过反复查阅源代码和实验手册后，我们发现我们没有对这两个变量进行赋初值的操作，这时我们意识到，static 变量需要在 cpp 文件或者头文件里初始化，即使其是类的变量。

五、 实验结论

本次实验相对来说较为简单，指导书上提供的框架也较为齐全，我们只要在原有的框架上进行一些改动即可。通过手动实现 MD5 算法，我们了解了其设计细节，并利用 C++ 高效的实现了该算法，提高了我们对信息安全领域的认知和兴趣。