网络安全技术

实
验
报
告

学　　院 计算机学院

年　　级 2017

班　　级 1 班

学　　号 1711425

姓　　名 曹元议

2020 年 6 月 9 日

目录

一、实验目的

1. 深入理解 MD5 算法的基本原理。
2. 掌握利用 MD5 算法生成数据摘要的所有计算过程。
3. 掌握检测文件完整性的基本方法。
4. 熟悉文件的基本操作方法。

二、实验内容

本次实验的总体要求是能够准确地实现 MD5 算法的完整计算过程，对于任意长度的字符串和任意大小的文件都能能够生成 128 位 MD5 摘要，并且能通过检查 MD5 摘要的正确性（通过从键盘输入或者从文件中获取旧的 MD5 摘要，和文件当前的 MD5 摘要进行比对）来检验原文件的完整性。
总而言之，程序的功能可由以下命令行参数表示：
`-h`：帮助选项
`-t`：计算一系列字符串的 MD5 值，测试程序的正确性
`-c [输入文件路径]`：计算指定的输入文件的 MD5 值
`-v [输入文件路径]`：从键盘输入文件路径指定文件的旧的摘要，程序再计算输入文件的新摘要，通过比较新旧摘要是否一致来判断文件是否被修改
`-f [输入文件路径] [.md5 文件路径]`：从指定的`.md5` 文件路径读取指定文件的旧的摘要，程序再计算输入文件的新摘要，通过比较新旧摘要是否一致来判断文件是否被修改

2.1 MD5 算法简述

MD5 算法是一种散列算法，是当前计算机领域为确保信息传输完整一致而广泛使用的散列算法之一，也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。
作为散列函数，MD5 有两个重要的特性：任意两组数据经过 MD5 运算后，生成相同摘要的概率微乎其微；即便在算法与程序已知的情况下，也不能够从 MD5 摘要中获得原始的数据。
MD5 可以为一个消息（例如文件、字符串）生成 128 位的哈希值（消息摘要）。

MD5 的一个典型应用就是为文件生成摘要，以校验它是否被修改。Linux 系统的`md5sum`命令可以为文件和字符串生成 MD5 摘要以及通过 MD5 摘要验证文件完整性。

MD5 算法的过程分为 3 个部分：消息的填充与分割、消息块的循环运算、摘要的生成。

2.1.1 消息的填充与分割

对于任意长度的输入数据，其 MD5 摘要的计算过程是以 512 bit 为一个分组，依次对数据的所有分组进行计算。因此首先需要让输入数据的比特长度为 512 的整数倍。消息填充的步骤如下：

1. 先判断文件（消息）的比特长度对 512 求余等于 448。
2. 如果满足上述条件，就在文件（消息）的末尾处添加 64 位（8 字节）的值，值的内容是原消息的长度（以位为单位）。
3. 如果大小（长度）不满足第 1 步中的条件，就执行以下操作：

(1) 填充 1 位 1

(2) 填充 0，直到满足第 1 步中的条件，再进行第 2 步

2.1.2 消息块的循环运算

MD5 算法中有四个 32 bit 的初始向量。它们分别是：$A = 0x01234567$，$B = 0x89abcdef$，$C = 0xfedcba98$，$D = 0x76543210$。由于考虑到内存数据大小端存储的问题，在程序中最好定义成（小端模式表示）：$A = 0x67452301$，$B = 0xefcdab89$，$C = 0x98badcfe$，$D = 0x10325476$。这些值仅在对该消息的第一个数据块作计算时才会用到。

在一次分块计算中，首先需要将初始向量 A、B、C、D 分别赋值给 4 个临时变量 a、b、c、d。

同时 MD5 算法规定了四个非线性操作函数如下（$\wedge$ 是按位与，$\vee$ 是按位或，$\neg$ 是按位取反，$\oplus$ 是按位异或）：

$$F(x, y, z) = (x \wedge y) \vee (\neg x \wedge z)$$

$$G(x, y, z) = (x \wedge z) \vee (y \wedge \neg z)$$

$$H(x, y, z) = x \oplus y \oplus z$$

$I(x,y,z)=y\oplus(x\vee\neg z)$

然后定义四个计算函数为（每个函数分别循环计算 16 次，共 64 次，其中 <<< 为循环左移，s 为循环左移的位数，$X[k]$ 为当前进行计算的数据块的 32 bit 消息子分组，k 为当前的子分组是该数据块的第 k 个子分组，常数 $T[i]$ 是 $4294967296\times|\sin(i)|$ 的整数部分）：

$FF(a,\ b,\ c,\ d,X[k],s,T[i])$ 表示

$$a=b+\left(\left(a+F(b,\ c,\ d)+X[k]+T[i]\right)<<<s\right),0\le i<16$$

$GG(a,\ b,\ c,\ d,X[k],s,T[i])$ 表示

$$a=b+\left(\left(a+G(b,\ c,\ d)+X[k]+T[i]\right)<<<s\right),16\le i<32$$

$HH(a,\ b,\ c,\ d,X[k],s,T[i])$ 表示

$$a=b+\left(\left(a+H(b,\ c,\ d)+X[k]+T[i]\right)<<<s\right),32\le i<48$$

$II(a,\ b,\ c,\ d,X[k],s,T[i])$ 表示

$$a=b+\left(\left(a+I(b,\ c,\ d)+X[k]+t[i]\right)<<<s\right),48\le i<64$$

上面的 4 轮共 64 次计算完成后，按照以下方式为初始向量 A、B、C、D 重新赋值：

$A=a+A$、$B=b+B$、$C=c+C$、$D=d+D$

对下一个数据块的运算的初始向量 A、B、C、D 的值将是在以上步骤中赋的新值。

2.1.3 摘要的生成

处理完所有的 512 位的分组后，得到一组新的 A、B、C、D 的值，将这些值按 $ABCD$ 的顺序级联，就得到了想要的 MD5 散列值。一般来说我们看到的 MD5 摘要字符串是以 16 进制的形式表示的。当然，输出依然要考虑内存存储的大小端问题（在本次实验中，程序中选择先将 A、B、C、D 的值转为大端再拼接）。

三、实验步骤及实验结果

本次实验的步骤为：编写代码→测试实验结果。
本次实验的编程部分主要分为两个步骤：MD5 算法的实现、程序功能及命令行交互逻辑的实现。为了实现的简单和方便起见，并且可以使用指针操作来增加本次实验的编程灵活性，本次实验使用 C++语言实现并且实现的功能完全参照实验文档的内容。

项目文件的源代码组织如下：
`MD5.h`：MD5 类型定义
`MD5.cpp`：MD5 类的函数及常量定义
`main.cpp`：主程序入口
`Makefile`：在 Linux 系统上编译并生成可执行程序的规则

为了使源码结构更加合理，将 MD5 算法的实现以及程序的功能函数都封装在类 `MD5` 中（在 `MD5.h` 中定义）：

```
1   class MD5
2   {
3   private:
4       /* 基本常量定义 */
5       static const uint32_t A;
6       static const uint32_t B;
7       static const uint32_t C;
8       static const uint32_t D;
9       static const uint32_t T[64];
10      static const uint32_t S[64];
11      static const uint32_t X[64];
12      static const string testString[];  // 用于验证程序正确性的测试字符串
13
14      /* 计算函数定义 */
15      uint32_t F(uint32_t x, uint32_t y, uint32_t z) {
16          return (x&y) | ((~x)&z);
17      }
18      uint32_t G(uint32_t x, uint32_t y, uint32_t z) {
19          return (x&z) | (y&(~z));
20      }
21      uint32_t H(uint32_t x, uint32_t y, uint32_t z) {
22          return x^y^z;
23      }
24      uint32_t I(uint32_t x, uint32_t y, uint32_t z) {
```

```
25          return y ^ (x | (~z));
26      }
27      void FF(uint32_t &a, uint32_t &b, uint32_t &c, uint32_t &d, uint32_t x,
  uint32_t i) {
28          uint32_t temp = F(b, c, d) + a + x + T[i];
29          temp = (temp << S[i]) | (temp >> (32 - S[i]));
30          a = b + temp;
31      }
32      void GG(uint32_t &a, uint32_t &b, uint32_t &c, uint32_t &d, uint32_t x,
  uint32_t i) {
33          uint32_t temp = G(b, c, d) + a + x + T[i];
34          temp = (temp << S[i]) | (temp >> (32 - S[i]));
35          a = b + temp;
36      }
37      void HH(uint32_t &a, uint32_t &b, uint32_t &c, uint32_t &d, uint32_t x,
  uint32_t i) {
38          uint32_t temp = H(b, c, d) + a + x + T[i];
39          temp = (temp << S[i]) | (temp >> (32 - S[i]));
40          a = b + temp;
41      }
42      void II(uint32_t &a, uint32_t &b, uint32_t &c, uint32_t &d, uint32_t x,
  uint32_t i) {
43          uint32_t temp = I(b, c, d) + a + x + T[i];
44          temp = (temp << S[i]) | (temp >> (32 - S[i]));
45          a = b + temp;
46      }
47
48      vector<uint8_t> data; // 要计算MD5值的字节流
49      uint32_t state[4];  // 保存加密结果的初始向量
50      string result;  // 加密结果字符串（16进制）
51
52      void calculateBlock(uint8_t *input); // 对字节流的一个分块（64字节）进
  行计算
53      void string2BitStream(string src); // string转字节流
54      void loadFile(char *filename); // 从文件中加载字节流
55      void padding();  // 按要求填充字节流
56      void compute();  // 计算填充后的字节流MD5值
57      void genResultString();  // MD5值转结果字符串
58      void computeString(string src); // 计算字符串的MD5值
59      void resetVector();  // 重置字节流、结果字符串、加密结果初始向量
60      bool judgeHexString(string src); // 判断是不是16进制字符串
61
62  public:
63      MD5();
```

```
64      void testApplication();  // 通过输出指定字符串的MD5值来验证程序正确性
65      void computeFile(char *filename, bool verbose = true); // 计算输入文件
  的MD5值
66      void validateFromString(char *srcFile);  // 通过手动输入的MD5值来校验
  程序是否被修改
67      void validateFromFile(char *md5File, char *srcFile); // 通过指定的MD5
  文件来校验程序是否被修改
68  };
```

其中，在 2.1.2 节中涉及到的一系列函数直接嵌入到类定义中，默认为内联函数。要计算的文件或字符串值都会放在 `uint8_t` 的 `vector` 类型的 `data` 变量中。`vector` 作为“动态数组”，使用非常方便，并且里面的所有元素的排列都是和数组的元素一样是连续存放的，通过取地址符&也能直接取到其中某元素所在的地址。

在 `MD5.cpp` 中，要定义程序中使用到的常量值：

```
17  const uint32_t MD5::A = 0x67452301;
18  const uint32_t MD5::B = 0xEFCDAB89;
19  const uint32_t MD5::C = 0x98BADCFE;
20  const uint32_t MD5::D = 0x10325476;
21
22  const string MD5::testString[] = { "","a","abc","message digest","abcdefghijklmnopqrstuvwxyz",
23  "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789",
24  "12345678901234567890123456789012345678901234567890123456789012345678901234567890" };
25
26  const uint32_t MD5::T[64] = {
27      0xD76AA478,0xE8C7B756,0x242070DB,0xC1BDCEEE,0xF57C0FAF,0x4787C62A,0xA8304613,0xFD469501,
28      0x698098D8,0x8B44F7AF,0xFFFF5BB1,0x895CD7BE,0x6B901122,0xFD987193,0xA679438E,0x49B40821,
29      0xF61E2562,0xC040B340,0x265E5A51,0xE9B6C7AA,0xD62F105D,0x02441453,0xD8A1E681,0xE7D3FBC8,
30      0x21E1CDE6,0xC33707D6,0xF4D50D87,0x455A14ED,0xA9E3E905,0xFCEFA3F8,0x676F02D9,0x8D2A4C8A,
31      0xFFFA3942,0x8771F681,0x6D9D6122,0xFDE5380C,0xA4BEEA44,0x4BDECFA9,0xF6BB4B60,0xBEBFBC70,
32      0x289B7EC6,0xEAA127FA,0xD4EF3085,0x04881D05,0xD9D4D039,0xE6DB99E5,0x1FA27CF8,0xC4AC5665,
33      0xF4292244,0x432AFF97,0xAB9423A7,0xFC93A039,0x655B59C3,0x8F0CCC92,0xFFEFF47D,0x85845DD1,
34      0x6FA87E4F,0xFE2CE6E0,0xA3014314,0x4E0811A1,0xF7537E82,0xBD3AF235,0x2AD7D2BB,0xEB86D391
35  };
36
37  const uint32_t MD5::S[64] = {
38      7,12,17,22,7,12,17,22,7,12,17,22,7,12,17,22,
39      5,9,14,20,5,9,14,20,5,9,14,20,5,9,14,20,
40      4,11,16,23,4,11,16,23,4,11,16,23,4,11,16,23,
41      6,10,15,21,6,10,15,21,6,10,15,21,6,10,15,21
42  };
43
44  const uint32_t MD5::X[64] = {
45      0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,
46      1,6,11,0,5,10,15,4,9,14,3,8,13,2,7,12,
47      5,8,11,14,1,4,7,10,13,0,3,6,9,12,15,2,
48      0,7,14,5,12,3,10,1,8,15,6,13,4,11,2,9
49  };
```

其中 `testString` 数组中的字符串是测试程序正确性子模块所使用到的测试字符串，其他都是 `MD5` 算法要使用到的常量（其中 `S` 数组为需要循环左移的位数，`X` 数组为该数据块的第几个 32 bit 子分组，其他常量的意义和 2.1.2 节中介绍的一

致）

3.1 MD5 算法的实现

3.1.1 源数据填充的实现

padding()函数对输入的源数据进行填充操作，按照 2.1.1 节的步骤将源数据长度填充为 512 位的整数倍。

```
1   void MD5::padding()
2   {
3       int fileBitSize = data.size() << 3;
4       int newSize = fileBitSize;
5       // 一直补充直到：比特流长度 % 512 == 448
6       if (newSize % 512 != 448)
7       {
8           data.push_back((uint8_t)0x80);
9           newSize += 8;
10          for (int i = newSize; newSize % 512 != 448; newSize += 8)
11              data.push_back(0);
12      }
13      // 补充 64 位文件长度
14      uint64_t fileBitSize64 = fileBitSize;
15      uint8_t *temp = (uint8_t*)&fileBitSize64;
16      for (int i = 0; i < 8; i++)
17      {
18          data.push_back(temp[i]);
19      }
20  }
```

3.1.2 消息块的循环运算的实现

首先，通过 compute()函数调用 calculateBlock()函数依次对填充后的源数据的每一个 512 位数据块进行循环计算：

```
1   void MD5::compute()
2   {
3       for (int i = 0; i < data.size(); i += 64)
4       {
```

```
5           calculateBlock(&data[i]);
6       }
7   }
```

calculateBlock()函数就是对每一个数据块进行 64 次计算的函数，步骤和方法与 2.1.2 节的内容一致。

```
1   void MD5::calculateBlock(uint8_t *input)
2   {
3       uint32_t a = state[0], b = state[1], c = state[2], d = state[3];
4       uint32_t *x = (uint32_t*)input; // 将64字节的数据块划分为16个4字节大
    小的子分组
5       int i = 0;
6       /* 共4轮运算，每一轮16次 */
7       /* 第1轮运算*/
8       for (; i < 16; i += 4)
9       {
10          FF(a, b, c, d, x[X[i]], i);
11          FF(d, a, b, c, x[X[i + 1]], i + 1);
12          FF(c, d, a, b, x[X[i + 2]], i + 2);
13          FF(b, c, d, a, x[X[i + 3]], i + 3);
14      }
15
16      /* 第2轮运算*/
17      for (; i < 32; i += 4)
18      {
19          GG(a, b, c, d, x[X[i]], i);
20          GG(d, a, b, c, x[X[i + 1]], i + 1);
21          GG(c, d, a, b, x[X[i + 2]], i + 2);
22          GG(b, c, d, a, x[X[i + 3]], i + 3);
23      }
24
25      /* 第3轮运算*/
26      for (; i < 48; i += 4)
27      {
28          HH(a, b, c, d, x[X[i]], i);
29          HH(d, a, b, c, x[X[i + 1]], i + 1);
30          HH(c, d, a, b, x[X[i + 2]], i + 2);
31          HH(b, c, d, a, x[X[i + 3]], i + 3);
32      }
33
34      /* 第4轮运算*/
35      for (; i < 64; i += 4)
36      {
37          II(a, b, c, d, x[X[i]], i);
```

```
38          II(d, a, b, c, x[X[i + 1]], i + 1);
39          II(c, d, a, b, x[X[i + 2]], i + 2);
40          II(b, c, d, a, x[X[i + 3]], i + 3);
41      }
42
43      // 将a、b、c、d 中的运算结果加到初始向量上
44      state[0] += a;  state[1] += b;  state[2] += c;  state[3] += d;
45  }
```

3.1.3 摘要字符串的生成

genResultString()函数将 state 数组的每一个元素转换成字符串再依次拼接，注意字节序的转换。

```
1   void MD5::genResultString()
2   {
3       for (int i = 0; i < 4; i++)
4       {
5           // trick: 由于先前的计算都是按照小端模式进行计算
6           // 因此转为字符串的时候要转换成大端模式
7           char temp[20] = { 0 };
8           sprintf(temp, "%08x", htonl(state[i]));
9           result += temp;
10      }
11  }
```

3.2 程序功能的实现

首先，loadFile()和 string2BitStream()函数分别将文件数据和字符串转换为字节流，并保存在 data 中。为了实现读取任何种类的文件（除了目录），需要将输入文件视为二进制文件：

```
1   void MD5::loadFile(char *filename)
2   {
3       assert(filename);
4       if (isDirectory(filename))
5       {
6           cout << filename << ": Is a directory" << endl;
7           exit(1);
8       }
9       ifstream infile(filename, ios::binary);
```

```
    if (!infile.is_open())
    {
        perror(filename);
        exit(1);
    }
    while (true)
    {
        uint8_t temp;
        infile.read((char*)&temp, sizeof(uint8_t)); // 逐字节读取文件
        if (infile.eof())
            break;
        data.push_back(temp);
    }
    infile.close();
}

void MD5::string2BitStream(string src)
{
    for (int i = 0; i < src.length(); i++)
    {
        data.push_back((uint8_t)src[i]);
    }
}
```

这里使用 `isDirectory()`函数来判断输入的文件是否是目录，如果是目录就终止程序。Linux 的 `md5sum` 也不能对一个目录计算 MD5 值。

```
bool isDirectory(char *filename)
{
    assert(filename);
    struct stat s;
    if (stat(filename, &s) == 0)
    {
        if (s.st_mode & S_IFDIR)
        {
            return true;
        }
    }
    return false;
}
```

`computeFile()`函数用于计算一个文件的 MD5 值，基于上述介绍的函数。如果

参数 verbose 的值为 true（默认情况下），则本次运行程序时计算完文件的 MD5 值以后会输出相应的信息并按照 Linux 的 `md5sum` 命令输出的格式输出.md5 文件（为了方便在 Windows 系统中测试）；否则 computeFile()函数将作为程序其他功能的子模块，不做信息打印和文件输出的操作。

```
1   void MD5::computeFile(char *filename, bool verbose)
2   {
3       resetVector();
4       loadFile(filename);
5       padding();
6       compute();
7       genResultString();
8       if (verbose)
9       {
10          cout << "The MD5 value of file(\"" << filename << "\") is:" << endl;
11          cout << result << endl;
12          // 将MD5 结果字符串输出到.md5 文件
13          string f(filename);
14          if (f.find('/') >= 0 || f.find('\\') >= 0)
15          {
16              int delim = max(f.find_last_of('/'), f.find_last_of('\\'));
17              f = f.substr(delim + 1);
18          }
19          if (f.find('.') >= 0)
20              f = f.substr(0, f.find_last_of('.'));
21          f += ".md5";
22          ofstream outfile(f);
23          outfile << result << "  " << filename << endl;
24          outfile.close();
25      }
26  }
```

testApplication()函数分别对数组 testString 的每一个字符串计算 MD5 值（通过 computeString()函数）并输出：

```
1   void MD5::testApplication()
2   {
3       int len = sizeof(testString) / sizeof(string);
4       for (int i = 0; i < len; i++)
5       {
6           computeString(testString[i]);
7       }
8   }
9
```

```
10  void MD5::computeString(string src)
11  {
12      resetVector();
13      string2BitStream(src);
14      padding();
15      compute();
16      genResultString();
17      cout << "MD5(\"" << src << "\") = " << result << endl;
18  }
```

注意，如果要计算下一个数据源的 MD5 值，需要通过 `resetVector()`函数将 `state` 数组的值复原以及清空 `data` 和 `result` 的内容，便于下次计算。

```
348  void MD5::resetVector()
349  {
350      data.clear();
351      result.clear();
352      state[0] = A;
353      state[1] = B;
354      state[2] = C;
355      state[3] = D;
356  }
```

`validateFromString()`函数的功能是通过从键盘中输入的字符串和指定文件的 MD5 值相比对，如果相同则表示文件未修改，否则表示文件已修改。这里会检查从键盘输入的字符串是否合法（可以是长度为 32 个字符的 16 进制字符串，大小写均可）。

```
1   void MD5::validateFromString(char *srcFile)
2   {
3       assert(srcFile);
4       if (isDirectory(srcFile))
5       {
6           cout << srcFile << ": Is a directory" << endl;
7           exit(1);
8       }
9       ifstream infile(srcFile);
10      if (!infile.is_open())
11      {
12          perror(srcFile);
13          exit(1);
14      }
15      infile.close();
16      string s;
17      do
```

```
18      {
19          if (s.length() > 0)
20              cout << "Error MD5 content format! Please input again!" << endl;
21          cout << "Please input the MD5 value of file(\"" << srcFile <<
    "\")... :" << endl;
22          getline(cin, s);
23          transform(s.begin(), s.end(), s.begin(), ::tolower);  // 转换成小
    写
24      } while (s.length() != 32 || !judgeHexString(s));
25      cout << "The old MD5 value of file(\"" << srcFile << "\") you have input
    is:" << endl;
26      cout << s << endl;
27      cout << "The new MD5 value of file(\"" << srcFile << "\") that has computed
    is:" << endl;
28      computeFile(srcFile, false);
29      cout << result << endl;
30      if (s == result)
31      {
32          cout << "OK! The file is integrated!" << endl;
33      }
34      else
35      {
36          cout << "Match Error! The file has been modified!" << endl;
37      }
38  }
```

其中用到的辅助函数 **judgeHexString()**用于判断一个字符串是否为 16 进制字符串。

```
1   bool MD5::judgeHexString(string src)
2   {
3       for (int i = 0; i < src.length(); i++)
4       {
5           if (isalpha(src[i]))
6           {
7               char c = tolower(src[i]);
8               if (c != 'a'&&c != 'b'&&c != 'c'&&c != 'd'&&c != 'e'&&c != 'f')
9                   return false;
10          }
11          else if (isdigit(src[i]))
12              continue;
13          else
14              return false;
```

```
    }
    return true;
}
```

validateFromFile()函数的功能是通过从指定的.md5 文件和指定源文件的 MD5 值相比对，如果相同则表示文件未修改，否则表示文件已修改。这里会检查文件的文本内容是否与 Linux 的 `md5sum` 命令的输出格式是否一致，文本中指定的文件名是否和命令行中指定的文件名是否一致，MD5 字符串的格式是否合法（长度为 32 个字符的 16 进制字符串）。

```
void MD5::validateFromFile(char *md5File, char *srcFile)
{
    assert(md5File && srcFile);
    if (isDirectory(srcFile))
    {
        cout << srcFile << ": Is a directory" << endl;
        exit(1);
    }
    if (isDirectory(md5File))
    {
        cout << md5File << ": Is a directory" << endl;
        exit(1);
    }
    ifstream infile(md5File);
    if (!infile.is_open())
    {
        perror(md5File);
        exit(1);
    }
    ifstream infile2(srcFile);
    if (!infile2.is_open())
    {
        perror(srcFile);
        exit(1);
    }
    infile2.close();
    string s, md5;
    while (getline(infile, s))
    {
        md5 += s;
    }
    infile.close();
    char md5_[50] = { 0 }, f[50] = { 0 };
    if (sscanf(md5.c_str(), "%s  %s", md5_, f) == 2)
```

```
35      {
36          string md5_str(md5_);
37          if (md5_str.length() != 32 || !judgeHexString(md5_str))
38          {
39              cout << "Error MD5 content format in " << md5File << "!" << endl;
40              exit(1);
41          }
42          if (strcmp(f, srcFile) != 0)
43          {
44              cout << f << ": Not match the source file name " << srcFile <<
    "!" << endl;
45              exit(1);
46          }
47      }
48      else
49      {
50          cout << "Error MD5 file content format in " << md5File << "!" << endl;
51          exit(1);
52      }
53      md5 = md5_;
54      transform(md5.begin(), md5.end(), md5.begin(), ::tolower);  // 转换成
    小写
55      cout << "The old MD5 value of file(\"" << srcFile << "\") in " << md5File
    << " is:" << endl;
56      cout << md5 << endl;
57      cout << "The new MD5 value of file(\"" << srcFile << "\") that has computed
    is:" << endl;
58      computeFile(srcFile, false);
59      cout << result << endl;
60      if (md5 == result)
61      {
62          cout << "OK! The file is integrated!" << endl;
63      }
64      else
65      {
66          cout << "Match Error! The file has been modified!" << endl;
67      }
68  }
```

最后，在 main.cpp 中，通过 parse_args()函数解析命令行参数并调用命令行参数指定的功能函数，其中 showUsage()函数为打印程序的帮助信息（如果命令行参数为-h、命令行参数格式有误或没有该参数指定的功能都会调用，然后退

出程序）：

```
#include "MD5.h"

MD5 md5;

void showUsage()
{
    cout << "MD5: usage: [-h]  --help information" << endl;
    cout << setw(12) << " " << "[-t]  --test MD5 application" << endl;
    cout << setw(12) << " " << "[-c]  [file path of the file computed]" << endl;
    cout << setw(20) << " " << "--compute MD5 of the given file" << endl;
    cout << setw(12) << " " << "[-v]  [file path of the file validated]" << endl;
    cout << setw(20) << " " << "--validate the integrality of a given file by manual input MD5 value" << endl;
    cout << setw(12) << " " << "[-f]  [file path of the file validated]  [file path of the .md5 file] " << endl;
    cout << setw(20) << " " << "--validate the integrality of a given file by read MD5 value from .md5 file" << endl;
    exit(1);
}

void parse_args(int argc, char *argv[])
{
    if (argc == 1)  // 不指定任何命令行参数
        showUsage();
    for (int i = 1; i < argc; )
    {
        if (strcmp(argv[i], "-h") == 0) // 帮助选项
        {
            showUsage();
        }
        else if (strcmp(argv[i], "-t") == 0)    // 测试字符串
        {
            if (i + 1 < argc)
            {
                showUsage();
            }
            else
            {
                md5.testApplication();
                break;
            }
```

```
39          }
40          else if (strcmp(argv[i], "-c") == 0) // 指定文件计算MD5值
41          {
42              if (i + 2 < argc)
43                  showUsage();
44              if (i + 1 < argc)
45              {
46                  md5.computeFile(argv[i + 1]);
47                  break;
48              }
49              else
50                  showUsage();
51          }
52          else if (strcmp(argv[i], "-v") == 0) // 输入字符串校验文件
53          {
54              if (i + 2 < argc)
55                  showUsage();
56              if (i + 1 < argc)
57              {
58                  md5.validateFromString(argv[i + 1]);
59                  break;
60              }
61              else
62                  showUsage();
63          }
64          else if (strcmp(argv[i], "-f") == 0) // 指定MD5文件校验文件
65          {
66              if (i + 3 < argc)
67                  showUsage();
68              else if (i + 2 < argc)
69              {
70                  md5.validateFromFile(argv[i + 2], argv[i + 1]);
71                  break;
72              }
73              else
74                  showUsage();
75          }
76          else
77          {
78              showUsage();
79          }
80      }
81  }
82
```

```
83  int main(int argc, char *argv[])
84  {
85      parse_args(argc, argv);
86      return 0;
87  }
```

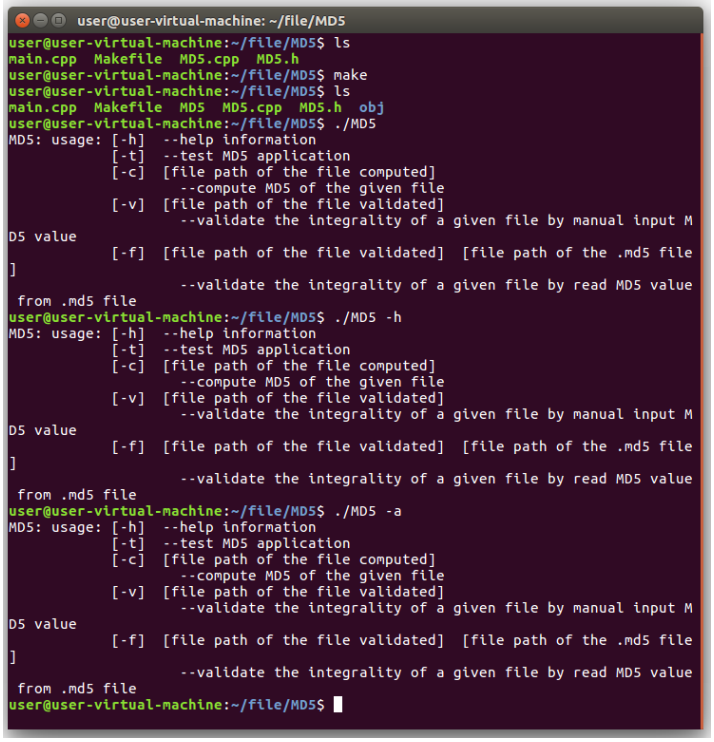
3.3 实验结果展示

3.3.1 运行程序的方法

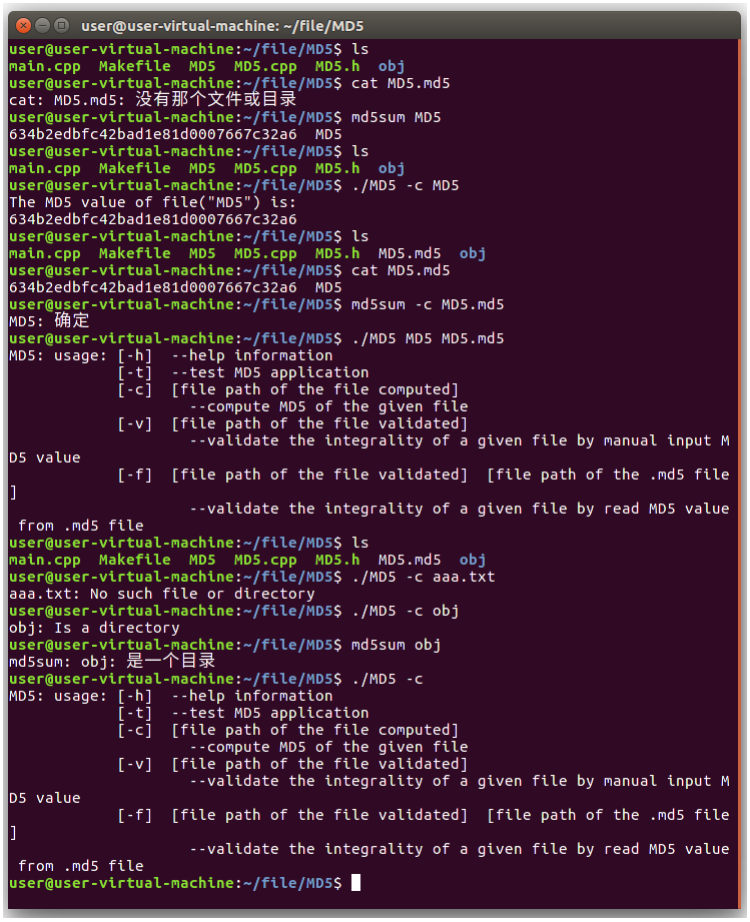
如果在 Linux 系统中，打开终端，定位到源码所在的目录下，直接输入 `make` 就可以编译并生成可执行文件（需要有 g++编译器）。
然后使用可执行文件并加上相应的命令行参数就可以运行程序。
如果在 Windows 系统中，打开 cmd 或 Powershell 或其他终端，定位到源码所在的目录下，直接使用可执行文件并加上相应的命令行参数即可运行程序。

3.3.2 样例测试

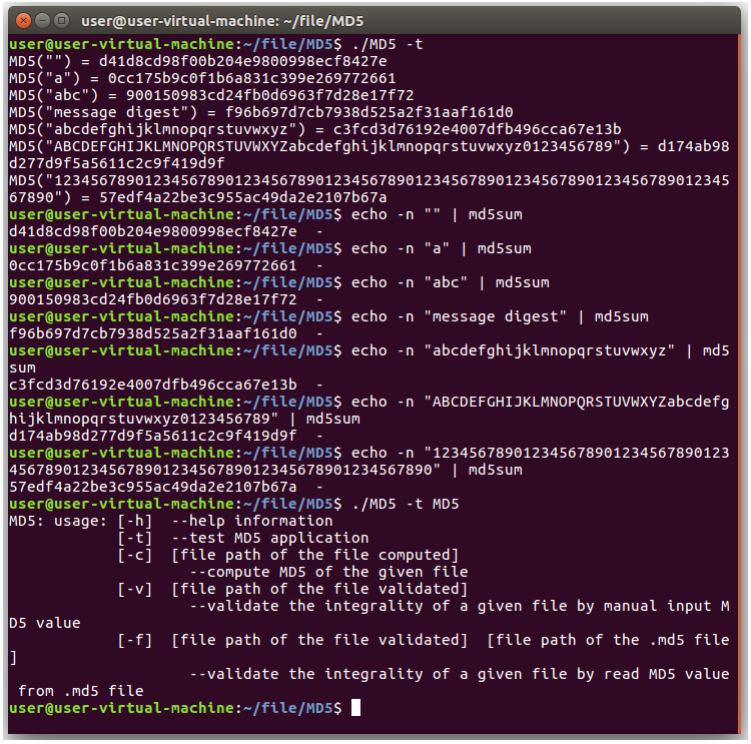
在 Ubuntu 16.04 系统中，在源码所在的目录下打开终端，并输入 `make` 命令即可编译并生成可执行文件。首先不指定命令行参数以及分别指定命令行参数为`-h` 和`-a` 的情况如下图所示。由于`-h` 选项是帮助选项，而不指定命令行参数或者指定`-a` 参数被视为命令行参数格式错误，所以都会显示帮助选项并退出程序。



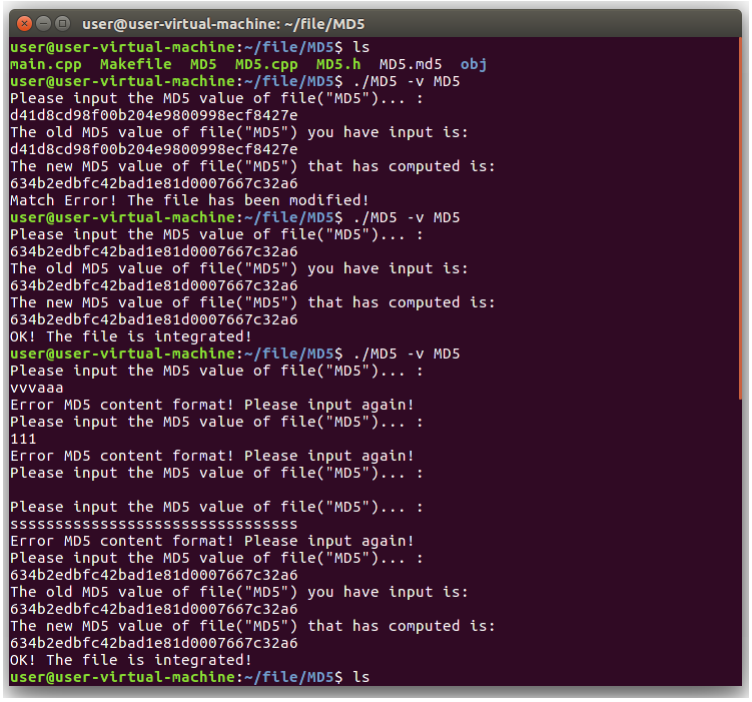
如果指定`-c`参数和输入文件参数（这里直接使用本次实验的可执行文件作为输入程序，是二进制文件），将会为该文件生成 MD5 摘要并输出到同名的`.md5`文件中。生成的 MD5 摘要和使用`md5sum`命令生成的 MD5 摘要一样。同时生成的`.md5`文件可以在`md5sum`程序中通过校验。如果指定的输入文件不存在，则会报文件不存在然后退出程序。如果指定的输入文件是目录，则会报这是个目录，然后退出程序。

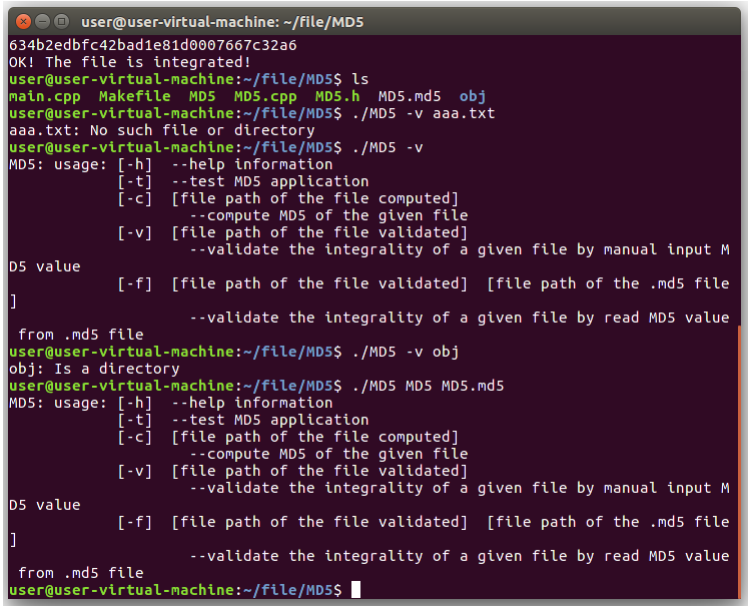


如果指定`-t`参数，则逐一测试`testString`数组中的字符串并输出相应的 MD5 字符串，和使用`md5sum`程序的输出一致。



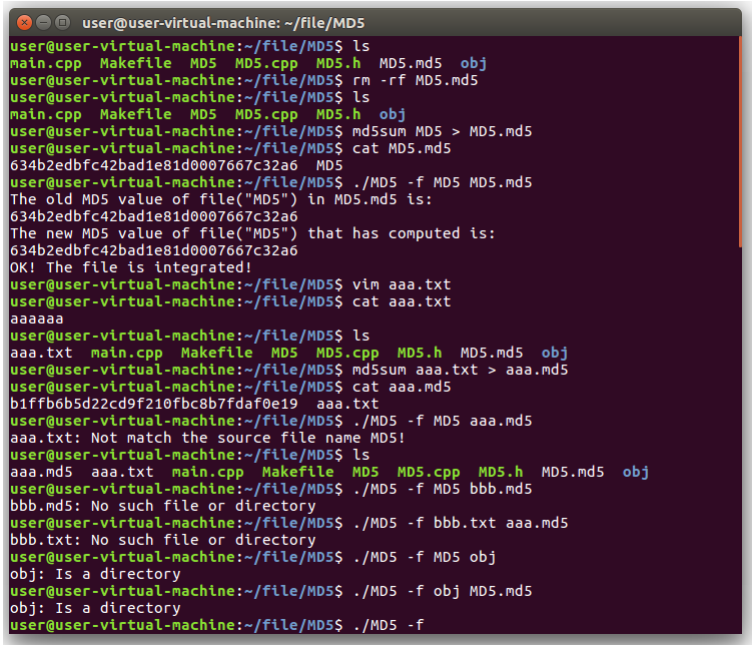
如果指定`-v`参数和相应的输入文件，则会让用户手动输入 MD5 字符串，程序会验证长度是否为 32 个字符并且字符串是否是 16 进制字符串，如果有一个条件不满足（没输入也算）则会让用户重新输入，直到输入的格式正确为止。

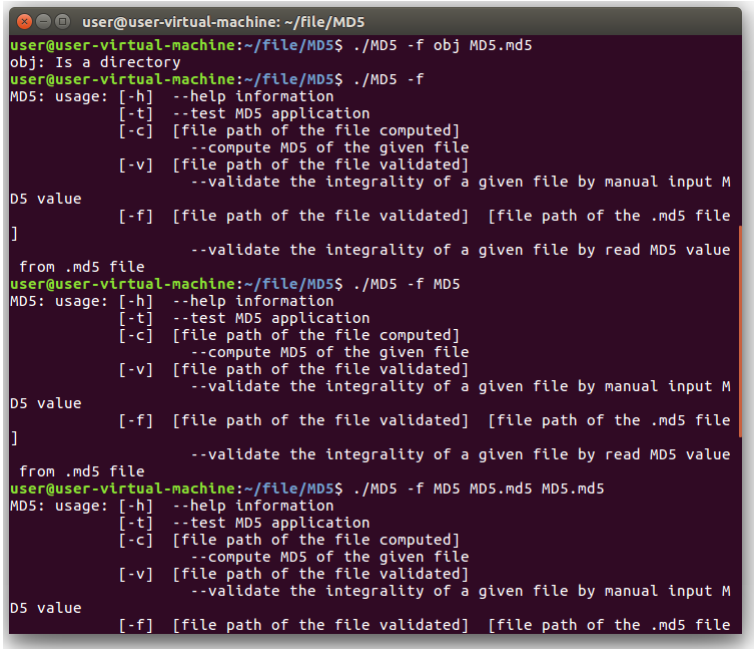


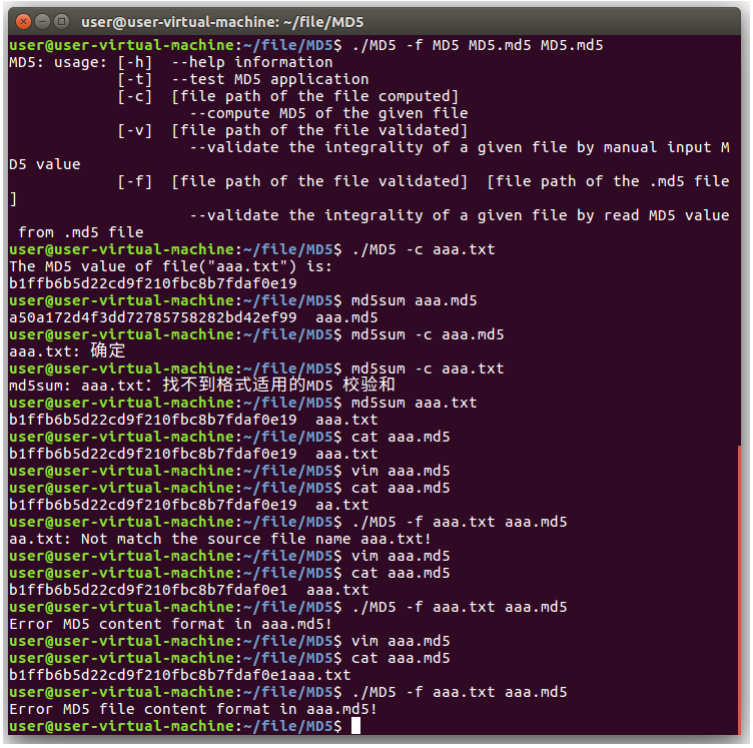


如果指定`-f`参数和相应的输入文件、`.md5`文件，则会从指定的`.md5`文件中读取 MD5 字符串，程序会验证长度是否为 32 个字符并且字符串是否是 16 进制字

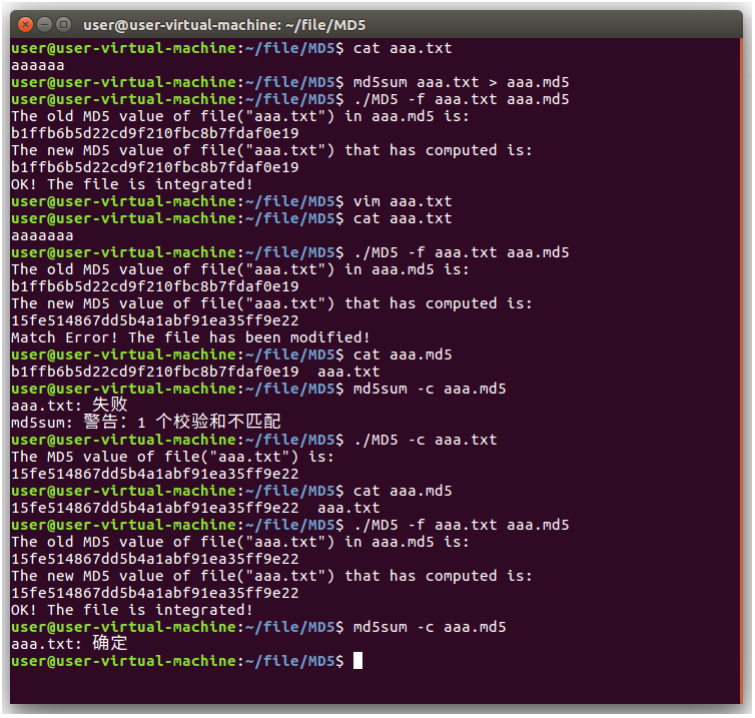
符串，以及 MD5 字符串右边所对应的文件和输入文件是否一致，如果有一个条件不满足程序会终止运行。如果输入的.md5 文件中的内容并不是“MD5 字符串文件路径”的格式，程序也会终止运行。







下图反映了对文本文件 `aaa.txt` 的结果也是类似的，也可以算出正确的 MD5 字符串，被修改之后则通不过程序的校验。



四、实验遇到的问题及其解决方法

这次实验遇到的最大的问题就是字节序转换的问题。最初在算完之后怎么运行都和标准的结果不一致，然后我的程序输出的结果的字节序刚好是反过来的，因为我原来都是按照小端模式去计算的。然后就想到了以前写的路由程序也有这个非常痛苦的过程，需要用 `htonl()`函数把 32 位无符号整数转换字节序为大端模式（代码见 3.1.3 节）。需要注意的是在 Windows 和 Linux 中 `htonl()`函数所属的头文件不一样，如果需要兼容不同的操作系统，需要根据不同的操作系统进行条件编译：

```
15  #ifdef _WIN32
16  #include <windows.h>
17  #pragma comment(lib, "wsock32.lib")
18  #else
19  #include <arpa/inet.h>
20  #endif
```

五、实验结论

1. 本次实验的程序生成的 MD5 字符串和 Linux 的 `md5sum` 命令生成的结果一样，可以说明本次实验的 MD5 算法实现正确。
2. 通过这次编程，更加深入地理解了 MD5 算法的原理，知道了 MD5 算法实际的用途。由于 MD5 和 SHA 都是生成信息摘要的算法，因此现在理解了为什么一些下载网站上会给出该文件的 SHA 散列值。由于网络不稳定等原因，下载下来的文件难免会有错误。这个就是来校验下载下来的文件是否和下载网站上提供的是一样的（通过校验下载下来的文件的 SHA 散列值是否和网站上的一致），否则就需要重新下载。