

南开大学

网络安全技术 课程实验报告

基于 MD5 算法的文件完整性校验程序



学院 网络空间安全学院

专业 信息安全

姓名 齐明杰

学号 2113997

2024 年 4 月 28 日

目 录

1	实验目的	4
2	实验内容	4
3	实验步骤	5
3.1	项目结构	5
3.2	MD5 算法实现	6
3.2.1	类封装	6
3.2.2	核心流程	7
3.2.2.1	初始化	8
3.2.2.2	更新状态	8
3.2.2.3	填充	12
3.2.2.4	输出	13
3.3	CLI 程序实现	14
3.3.1	main 函数	15
3.3.2	帮助信息函数 showHelp	16
3.3.3	测试 MD5 函数 testMD5	17
3.3.4	计算文件 MD5 computeMD5	18
3.3.5	手动验证 MD5 validateMD5Manual	18
3.3.6	通过.md5 文件验证 MD5 validateMD5File	19
3.3.7	总结	20
3.4	程序编译	21
3.5	运行结果	22
3.5.1	显示帮助信息	22
3.5.2	测试 MD5 算法	23
3.5.3	计算文件的 MD5 哈希值	23
3.5.4	手动验证文件的 MD5 哈希值	23
3.5.5	通过 .md5 文件验证文件的 MD5 哈希值	24
4	实验遇到的问题及解决方法	24

4.1	问题一：MD5 的大小端问题	24
4.2	问题二：命令程序的参数	25
4.3	问题三：文件读取的异常处理	26
5	实验结论	27

图表

图 3.1.1:	项目结构	5
表 3.1.2:	各个代码功能	6
图 3.2.3:	对数据块分块进行运算	9
图 3.2.4:	transform 变换过程	11
图 3.2.5:	填充比特	12
表 3.3.6:	各个命令功能	15
图 3.4.7:	执行 make 编译	22
图 3.4.8:	生成的可执行文件	22
图 3.5.9:	帮助信息	23
图 3.5.10:	测试 MD5 算法	23
图 3.5.11:	计算文件 HASH	23
图 3.5.12:	输入 HASH 进行文件校验	24
图 3.5.13:	创建.md5 文件	24
图 3.5.14:	利用.md5 进行文件校验	24
图 4.3.15:	找不到文件处理	27

1 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。

i 训练目的与要求

本章编程训练的目的如下：

- 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 Linux 系统中检测文件完整性的基本方法。
- ④ 熟悉 Linux 系统中文件的基本操作方法。

本章编程训练的要求如下：

- ① 准确地实现 MD5 算法的完整计算过程。
- ② 对于任意长度的字符串能够生成 128 位 MD5 摘要。
- ③ 对于任意大小的文件能够生成 128 位 MD5 摘要。
- ④ 通过检查 MD5 摘要的正确性来检验原文件的完整性。

2 实验内容

编程练习要求

在 Linux 平台下编写应用程序，正确地实现 MD5 算法。要求程序不仅能够为任意长度的字符串生成 MD5 摘要，而且可以为任意大小的文件生成 MD5 摘要。同时，程序还可以利用 MD5 摘要验证文件的完整性。验证文件完整性分为两种方式：一种是在手动输入 MD5 摘要的条件下，计算出当前被测文件的 MD5 摘要，再将两者进行比对。若相同，则文件完好；否则，文件遭到破坏。另一种是先利用 Linux 系统工具 md5sum 为被测文件生成一个后缀为.md5 的同名文件，然后让程序计算出被测文件的 MD5 摘要，将其与.md5 文件中的 MD5 摘要进行比较，最后得出检测结果。具体要求如下所示。

程序输入格式

程序为命程序，可执行文件名为 MD5.exe，命令行格式如下：

`./MD5 [选项] [被测文件路径] [.md5 文件路径]`

其中[选项]是程序为用户提供的各种功能。在本程序中[选项]包括{-h,-t,-c,-v,-f}5 个基本功能。[被测文件路径]为应用程序指明被测文件在文件系统中的路径。[.md5 文件路径]为

应用程序指明由被测文件生成的.md5 文件在文件系统中的路径。其中前两项为必选项，后两项可以根据功能进行选择。

程序执行过程

(1) 打印帮助信息

在控制台命令行中输入 `./MD5 -h`，打印程序的帮助信息。帮助信息详细地说明了程序的选项和执行参数。用户可以通过查询帮助信息充分了解程序的功能。

(2) 打印测试信息

在控制台命令行中输入 `./MD5 -t`，打印程序的测试信息。如下所示，测试信息是指本程序对特定字符串输入所生成的 MD5 摘要。所谓特定的字符串是指在 MD5 算法官方文档（RFC1321）中给出的字符串。同时，该文档也给出了这些特定字符串的 MD5 摘要。因此我们只需要将本程序的计算结果与文档中的正确摘要进行比较，就可以验证程序的正确性。

(3) 为指定文件生成 MD5 摘要

在控制台命令行中输入 `./MD5 -c [被测文件路径]`，计算出的被测文件的 MD5 摘要并打印出来。被测文件 `nankai.txt` 与可执行文件 `MD5` 处于同一个目录中。

(4) 验证文件完整性方法一

在控制台命令行中输入 `./MD5 -c [被测文件路径]`，程序会先让用户输入被测文件的 MD5 摘要，然后重新计算被测文件的 MD5 摘要，最后将两个摘要逐位比较。若一致，则说明文件是完整的，否则，说明文件遭到破坏。

(5) 验证文件完整性方法二

在控制台命令行输入 `./MD5 -f [被测文件路径] [.md5 文件路径]`，程序会自动读取.md5 文件中的摘要，然后重新计算出被测文件的 MD5 摘要，最后将两者逐位比较。若一致，则说明文件是完整的，否则，说明文件遭到破坏。

3 实验步骤

3.1 项目结构

本次实验需要实现一个基于 MD5 算法的文件完整性校验程序，主要包括 MD5 算法的实现和 CLI 程序的设计。为了更好地组织代码和功能，我将项目分为以下几个模块：

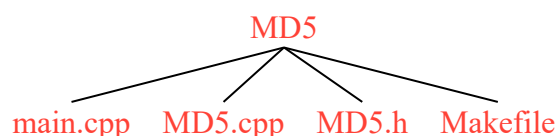


图 3.1.1: 项目结构

文件	功能
MD5.h	MD5 类的声明
MD5.cpp	MD5 类的实现
main.cpp	CLI 程序的入口
Makefile	编译配置文件

表 3.1.2: 各个代码功能

3.2 MD5 算法实现

3.2.1 类封装

采用面向对象的方式实现 MD5 算法, 将 MD5 算法封装为一个类, 方便调用和管理。MD5 类的声明和实现分别在 MD5.h 和 MD5.cpp 文件中, 类的声明如下:

```
1  class MD5 {
2  public:
3      MD5();
4      ~MD5();
5
6      // MD5 接口
7      std::string computeMD5(const unsigned char* input, size_t length);
8      std::string computeMD5(const std::string& input);
9
10     // 重置 MD5 状态, 准备新的计算
11     void reset();
12
13 private:
14     // MD5 基本变换过程, 处理一个 64 字节块
15     void transform(const unsigned char block[64]);
16
17     // 用于更新状态的函数, 可以多次调用
18     void update(const unsigned char* input, size_t length);
19     void update(const std::string& input);
20
21     // 完成摘要计算
22     std::vector<unsigned char> finalize();
23
24     // 返回十六进制格式的摘要
25     std::string toHexString();
26
27     // 编码和解码函数, 用于操作字节和 32 位数之间的转换
28     static void encode(uint32_t input[], unsigned char output[], size_t len);
```

```

29     static void decode(const unsigned char input[], uint32_t output[], size_t
len);
30
31     // 用于 MD5 计算的辅助函数
32     static uint32_t F(uint32_t x, uint32_t y, uint32_t z);
33     static uint32_t G(uint32_t x, uint32_t y, uint32_t z);
34     static uint32_t H(uint32_t x, uint32_t y, uint32_t z);
35     static uint32_t I(uint32_t x, uint32_t y, uint32_t z);
36     static void FF(uint32_t& a, uint32_t b, uint32_t c, uint32_t d, uint32_t
x, uint32_t s, uint32_t ac);
37     static void GG(uint32_t& a, uint32_t b, uint32_t c, uint32_t d, uint32_t
x, uint32_t s, uint32_t ac);
38     static void HH(uint32_t& a, uint32_t b, uint32_t c, uint32_t d, uint32_t
x, uint32_t s, uint32_t ac);
39     static void II(uint32_t& a, uint32_t b, uint32_t c, uint32_t d, uint32_t
x, uint32_t s, uint32_t ac);
40
41     // 循环左移
42     static uint32_t rotate_left(uint32_t x, int n);
43
44     // MD5 上下文变量
45     uint32_t state[4]; // MD5 状态(A, B, C, D)
46     uint32_t count[2]; // 位数计数器
47     unsigned char buffer[64]; // 数据输入缓冲区
48     std::vector<unsigned char> digest; // 存储最终摘要的数组
49     bool finalized; // 表示摘要是否已经生成
50
51     static const unsigned char PADDING[64]; // 填充用的数组
52 };

```

MD5 的核心算法主要包括四个步骤:初始化、填充、处理、输出。MD5 类中的 computeMD5 函数用于计算给定输入的 MD5 哈希值,它可以接受字符串或字节数组作为输入,并返回计算得到的 MD5 哈希值。MD5 类还提供了 reset 函数,用于重置 MD5 状态,以便进行新的计算。MD5 类的实现主要包含了 MD5 算法的核心变换过程、状态更新、摘要计算和编码解码等功能,确保了 MD5 算法的正确性和高效性。

3.2.2 核心流程

接下来让我们顺着 MD5 算法的核心流程,来看一下 MD5 类的实现。MD5 算法主要包括四个步骤:初始化、填充、处理和输出。MD5 类的实现中,transform 函数用于处理一个 64 字节块的数据,update 函数用于更新 MD5 状态,finalize 函数用于完成摘要计算,toHexString 函数用于将摘要转换为十六进制格式的字符串。

3.2.2.1 初始化

初始化步骤是至关重要的第一阶段，它设定了算法处理数据所需的初始环境和参数。

```
1  // 构造函数
2  MD5::MD5() : finalized(false) {
3      memset(buffer, 0, sizeof(buffer));
4      count[0] = count[1] = 0;
5      // 初始化 MD5 的四个主要数据寄存器
6      state[0] = 0x67452301;
7      state[1] = 0xEFCDAB89;
8      state[2] = 0x98BADCFE;
9      state[3] = 0x10325476;
10 }
```

在构造函数中，finalized 标志首先被设置为 false，表示 MD5 计算过程尚未完成。此外，输入缓冲区 buffer 被清零，以便存储即将处理的数据块。count 数组，负责追踪处理的数据总位数，也被初始化为零。最关键的初始化步骤是设定 MD5 算法的四个核心状态寄存器 state[0] 到 state[3]，这四个值不是随机的，而是根据 MD5 算法的标准规范设定的固定值，这些值是算法正确执行的基础。

```
1  // MD5 初始化函数，初始化核心变量
2  void MD5::reset() {
3      finalized = false;
4      count[0] = count[1] = 0;
5      state[0] = 0x67452301;
6      state[1] = 0xEFCDAB89;
7      state[2] = 0x98BADCFE;
8      state[3] = 0x10325476;
9      digest.clear();
10 }
```

reset 方法提供了一种重新初始化 MD5 对象的方式，无论何时调用，都将 MD5 状态寄存器、计数器和 finalized 标志重置到初始状态。这在多次使用同一个 MD5 对象进行不同数据计算时非常有用。此外，digest 向量，用于存储最终的哈希结果，也被清除，确保每次开始新的 MD5 计算时，环境都是干净的。

这种初始化保证了 MD5 算法可以从一个预定义的、一致的状态开始处理数据，无论其被用来计算哪种数据的 MD5 值。这对于保持算法的可靠性和预测性是必要的，也是所有加密哈希函数共有的重要特性。

3.2.2.2 更新状态

在 MD5 算法的核心流程中，“更新状态”步骤是用来处理输入数据，并更新内部状态的关

键环节。这一步骤负责将输入数据分块处理，确保每个数据块都能正确地影响最终的哈希值。

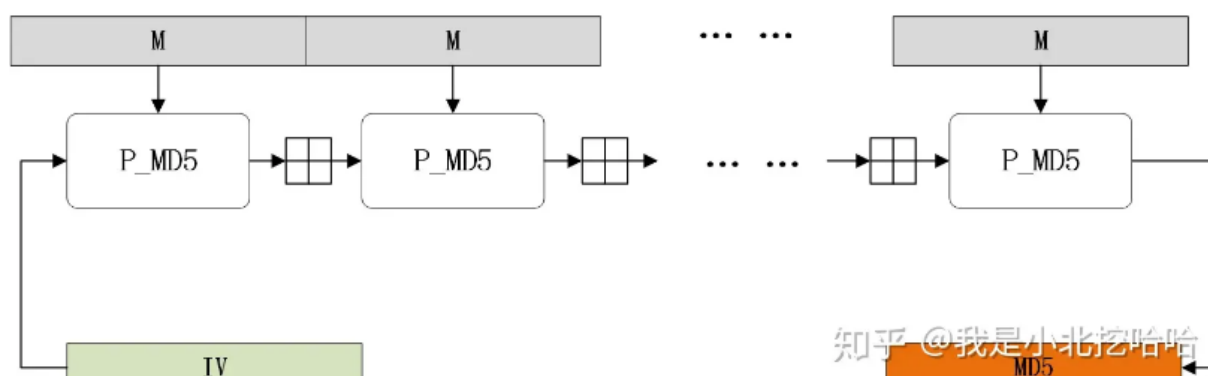


图 3.2.3: 对数据块分块进行运算

```

1  // 更新 MD5, 输入是原始的数据和长度
2  void MD5::update(const unsigned char* input, size_t length) {
3      size_t index = count[0] / 8 % 64;
4      size_t partLen = 64 - index;
5      size_t i = 0;
6
7      // 更新位数计数
8      count[0] += (uint32_t)(length << 3);
9      if (count[0] < (length << 3))
10         count[1]++;
11     count[1] += (uint32_t)(length >> 29);
12
13     // 足够填满一个 64 字节块
14     if (length >= partLen) {
15         memcpy(&buffer[index], input, partLen);
16         transform(buffer);
17
18         for (i = partLen; i + 63 < length; i += 64) {
19             transform(&input[i]);
20         }
21         index = 0;
22     }
23
24     // 输入剩余部分
25     memcpy(&buffer[index], &input[i], length - i);
26 }
  
```

update 函数的作用是将输入的数据分块，每块 64 字节，并逐块进行处理。函数首先计算出当前缓冲区 buffer 中未填满部分的起始位置和长度。然后，它更新数据的位数计

数器 count，这是因为 MD5 算法的输出依赖于整个消息的位长度。这里使用了位运算来确保即使数据长度非常大，位计数也能正确增加。

当输入数据足以填满至少一个块时，update 方法将数据复制到缓冲区并调用 transform 函数处理这个块。这个过程会在输入数据足够多的情况下重复进行，直到所有数据都被处理。剩余的数据（不足以构成一个完整的块的数据）会保留在缓冲区中，等待下一次调用 update 或最终的 finalize 函数时一并处理。

```
1  // MD5 辅助函数实现
2  uint32_t MD5::F(uint32_t x, uint32_t y, uint32_t z) { return (x & y) | (~x
3  & z); }
4  uint32_t MD5::G(uint32_t x, uint32_t y, uint32_t z) { return (x & z) | (y
5  & ~z); }
6  uint32_t MD5::H(uint32_t x, uint32_t y, uint32_t z) { return x ^ y ^ z; }
7  uint32_t MD5::I(uint32_t x, uint32_t y, uint32_t z) { return y ^ (x | ~z); }
8  uint32_t MD5::rotate_left(uint32_t x, int n) {
9      return (x << n) | (x >> (32 - n));
10 }
11
12 void MD5::FF(uint32_t& a, uint32_t b, uint32_t c, uint32_t d, uint32_t x,
13 uint32_t s, uint32_t ac) {
14     a += F(b, c, d) + x + ac;
15     a = rotate_left(a, s) + b;
16 }
17
18 void MD5::GG(uint32_t& a, uint32_t b, uint32_t c, uint32_t d, uint32_t x,
19 uint32_t s, uint32_t ac) {
20     a += G(b, c, d) + x + ac;
21     a = rotate_left(a, s) + b;
22 }
23
24 void MD5::HH(uint32_t& a, uint32_t b, uint32_t c, uint32_t d, uint32_t x,
25 uint32_t s, uint32_t ac) {
26     a += H(b, c, d) + x + ac;
27     a = rotate_left(a, s) + b;
28 }
```

这些辅助函数是 MD5 算法中四轮变换的核心，它们根据不同的逻辑关系和常数来更新 MD5 算法的状态变量。这些函数的实现是 MD5 算法正确性的关键，它们确保了 MD5 算法对输入数据的每一位都能产生影响，从而生成唯一的哈希值。这些函数的实现是 MD5 算法的核心，也是 MD5 类的重要组成部分。

```

1  // MD5 的主要变换程序
2  void MD5::transform(const unsigned char block[64]) {
3      uint32_t a = state[0], b = state[1], c = state[2], d = state[3], x[16];
4      decode(block, x, 64);
5      // 各轮变换逻辑 (FF, GG, HH, II 轮函数调用)
6      ...
7      // 状态更新
8      state[0] += a;
9      state[1] += b;
10     state[2] += c;
11     state[3] += d;
12     // 清理
13     memset(x, 0, sizeof(x));
14 }
15

```

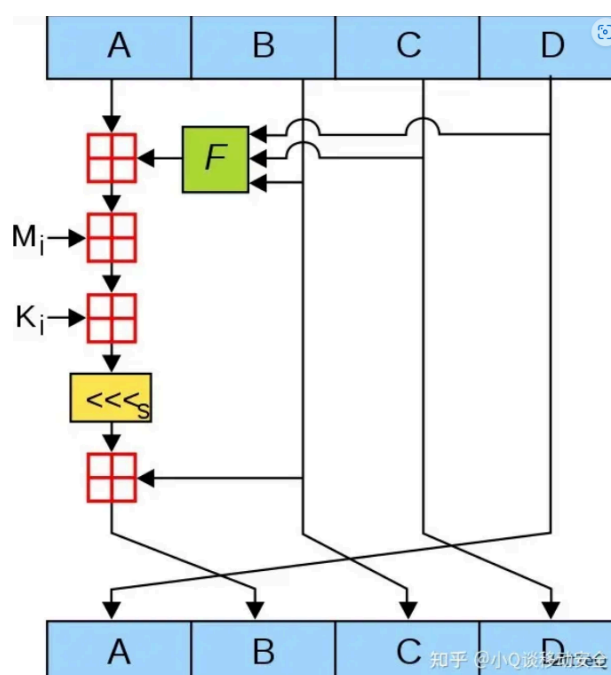


图 3.2.4: transform 变换过程

transform 函数是 MD5 算法的核心，负责对单个 64 字节块执行复杂的加密变换。该函数首先从块中解码出 16 个 32 位的子块，然后使用这些子块通过四轮的特定制密运算（如 FF、GG、HH、II）来更新算法的状态变量。每一轮包括多步的特定制密运算，这些运算

依赖于不同的辅助函数 (F, G, H, I) 和预定义的常数。这些运算确保了数据的每一位都能影响最终的哈希值，增强了算法对原始数据微小变化的敏感性。

在完成所有变换后，transform 方法将处理结果累加回 MD5 的状态变量中，为下一个块的处理或最终的摘要计算做准备。这种设计使得 MD5 算法不仅能够处理任意长度的数据，而且保证了算法的安全性和效率。

3.2.2.3 填充

在 MD5 算法中，填充是必要的步骤之一，用于确保处理的数据长度是 512 位 (64 字节) 的整数倍。这一步骤对于生成正确的哈希值至关重要，因为 MD5 算法的设计要求输入数据必须通过特定方式扩展，以便在最后的数块中还能包含原始数据的长度信息。

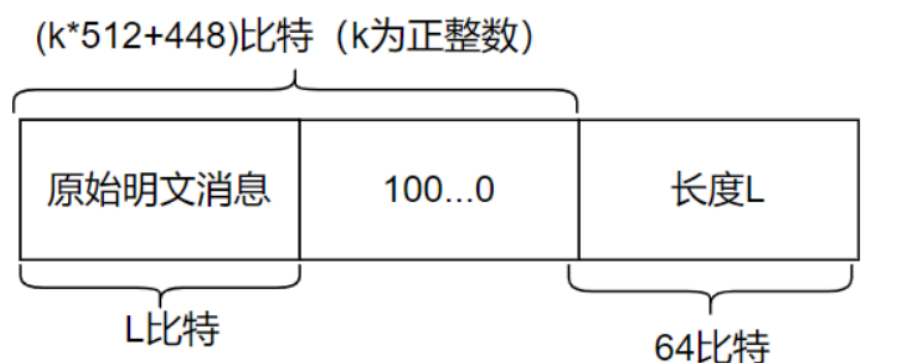


图 3.2.5: 填充比特

```
1 // 完成 MD5 计算，返回摘要
2 std::vector<unsigned char> MD5::finalize() {
3     unsigned char bits[8];
4     size_t index, padLen;
5
6     if (finalized)
7         return digest;
8
9     // 保存位数
10    encode(count, bits, 8);
11
12    // 填充到 56 字节
13    index = (uint32_t)((count[0] >> 3) & 0x3f);
14    padLen = (index < 56) ? (56 - index) : (120 - index);
15    update(PADDING, padLen);
16
17    // 加上位数
18    update(bits, 8);
19
20    // 存储状态到摘要
```

cpp

```
21  digest.resize(16);
22  encode(state, &digest[0], 16);
23
24  // 清理变量
25  memset(buffer, 0, sizeof(buffer));
26  memset(count, 0, sizeof(count));
27  finalized = true;
28
29  return digest;
30 }
```

finalize 函数的核心目的是完成数据的填充, 确保每条消息在处理前都扩展到恰当的长度。具体步骤如下:

1. **保存位数:** 首先, 函数将已处理的数据的总位数编码到一个 8 字节的数组 bits 中。这是为了确保最终的数据块可以包含原始数据长度的信息, 这一信息是生成最终哈希值的关键部分。
2. **计算填充长度:** 算法计算还需要填充多少字节才能使数据长度达到 56 字节模 64 字节 (因为最后的 8 字节用于存放数据长度)。这是通过计算已经填充的数据长度并与 56 取差值实现的。
3. **添加填充数据:** 使用标准的填充模式 (通常是一个 0x80 跟随若干 0x00), 确保数据的长度满足 MD5 处理的需求。这种填充方式确保了无论原始数据的长度如何, 填充后的数据都能正确地被算法处理。
4. **添加原始数据长度:** 填充之后, 原始数据的长度 (以位为单位) 被添加到数据的末尾, 这使得数据的总长度现在是 64 字节的倍数。
5. **生成最终摘要:** 完成填充后, 使用 encode 函数将内部状态转换为最终的摘要值。这些值被存储在 digest 向量中, 之后返回。

3.2.2.4 输出

在 MD5 算法的实现中, 输出阶段负责将最终的哈希值转换为一种可读的格式, 通常是十六进制字符串。这一步是用户获取和利用生成的哈希值的关键环节。

```
1  // 十六进制转换
2  std::string MD5::toHexString() {
3      if (!finalized)
4          return "";
5
6      char buf[33];
7      for (int i = 0; i < 16; i++)
8          sprintf(buf + i * 2, "%02x", digest[i]);
9      buf[32] = 0;
```

cpp

```
10
11     return std::string(buf);
12 }
```

toHexString 函数的主要作用是将内部保存的二进制哈希摘要转换为易于阅读和存储的十六进制字符串格式。这个转换过程涉及以下几个关键步骤：

1. **检查摘要状态**：首先检查 finalized 标志，确保在尝试生成十六进制字符串之前，摘要计算已经完成。如果 finalized 为 false，表示哈希摘要尚未准备好，函数将返回一个空字符串。
2. **格式化字符串**：使用 sprintf 函数，将每个字节的二进制哈希值转换为其对应的两个字符的十六进制表示。这里使用 %02x 格式指示符确保每个字节均格式化为两位十六进制数，如果数值小于 16，前面会补一个零。
3. **字符串构建**：字符数组 buf 被初始化为长度为 33 的数组，足够存放 16 个字节的十六进制表示（32 字符）和一个字符串结束符。循环通过 sprintf 依次填充这个数组，然后在末尾手动设置字符串结束符，确保字符串正确结束。

```
1 // 处理 std::string 数据
2 std::string MD5::computeMD5(const std::string& input) {
3     reset(); // 重置 MD5 对象状态
4     update(input); // 更新状态，使用 std::string 重载的 update
5     finalize(); // 完成哈希计算
6     return toHexString(); // 返回十六进制字符串
7 }
```

另外我还提供了一个 computeMD5 函数，用于处理 std::string 类型的数据。这个函数首先调用 reset 方法重置 MD5 对象的状态，然后调用 update 方法更新状态，使用 std::string 类型的数据作为输入。最后，调用 finalize 方法完成哈希计算，并调用 toHexString 方法将结果转换为十六进制字符串。这个函数提供了一种方便的方式，让用户可以直接使用字符串数据进行 MD5 计算。

3.3 CLI 程序实现

” CLI

CLI，即命令行界面（Command Line Interface），是一种允许用户通过文本命令与计算机程序进行交互的接口。CLI 通常用于在终端或控制台环境中执行命令，以便操作系统或应用程序执行特定的任务。CLI 提供了一种直接而高效的方式来与计算机进行交互，尤其适用于需要快速执行命令或脚本的场景。

在本实验作业中，我设计并实现了一个命令行界面（CLI），以使用户通过命令行与 MD5 哈希算法交互。此 CLI 是程序的主要用户界面，允许用户在命令行环境中执行各种操作，如计算字符串或文件的 MD5 哈希值，以及验证文件的完整性与给定的 MD5 哈希值是否匹配。

CLI 的实现主要包括以下几个关键部分：

- 1. **参数解析**：程序通过命令行参数接收用户输入，这些参数指定了程序应执行的具体操作。实现了参数解析功能，能够识别和处理用户输入的不同命令和选项。
- 2. **功能执行**：根据用户输入的参数，程序选择相应的操作模式。例如，用户可以指定生成一个文本或文件的 MD5 哈希值，或者对比文件哈希值以验证其完整性。
- 3. **交互反馈**：为了提升用户体验，CLI 提供了实时反馈，包括操作指引、执行结果和错误消息。这确保用户能够明白操作的结果，以及如何在出现问题时进行纠正。
- 4. **错误处理**：程序能够优雅地处理错误情况，比如非法的命令行输入或文件读取错误。在这些情况下，CLI 会输出适当的错误信息，并指导用户如何正确使用工具。

实现这样的命令行界面，不仅使得实验工具易于使用且功能强大，而且还确保了程序能够适应多种不同的操作环境，包括自动化脚本和手动终端操作。这种灵活性和易用性是本实验中特别关注的目标。

CLI 程序实现

命令	功能
-h	显示帮助信息
-t	测试 MD5 算法
-c [file path]	计算指定文件的 MD5
-v [file path]	手动验证文件的 MD5
-f [file path] [.md5 file path]	通过.md5 文件验证文件的 MD5

表 3.3.6: 各个命令功能

3.3.1 main 函数

```
1  int main(int argc, char* argv[]) {
2      // 检查是否有足够的命令行参数
3      if (argc < 2) {
4          showHelp(); // 如果没有足够的参数，显示帮助信息
5          return 1;
6      }
7      string option = argv[1]; // 获取第一个命令行参数作为选项
8      // 根据不同的命令行选项执行不同的功能
9      if (option == "-h") {
10         showHelp();
```



```
11     }
12     else if (option == "-t") {
13         testMD5();
14     }
15     else if (option == "-c" && argc == 3) {
16         computeMD5(argv[2]); // 计算指定文件的 MD5
17     }
18     else if (option == "-v" && argc == 3) {
19         validateMD5Manual(argv[2]); // 手动验证文件的 MD5
20     }
21     else if (option == "-f" && argc == 4) {
22         validateMD5File(argv[2], argv[3]); // 通过 .md5 文件验证文件的 MD5
23     }
24     else {
25         cout << "Invalid option or missing arguments\n";
26         showHelp(); // 如果输入的命令行选项无效，显示帮助信息
27         return 1;
28     }
29     return 0;
30 }
```

解析

此段代码是程序的入口，它首先检查用户是否输入了必要的命令行参数。如果没有输入足够的参数，程序将调用 `showHelp` 函数来显示如何正确使用程序的说明。根据用户输入的选项（如 `-h`, `-t`, `-c`, `-v`, `-f`），程序将执行不同的函数来进行 MD5 的测试、计算或验证。

3.3.2 帮助信息函数 `showHelp`

```
1 void showHelp() {
2     cout << "MD5: usage: [-h] --help information\n"
3         << "[-t] --test MD5 application\n"
4         << "[-c] [file path] --compute MD5 of the given file\n"
5         << "[-v] [file path] --validate the integrity of a given file by manual
   input MD5 value\n"
6         << "[-f] [file path of the file validated] [file path of the .md5 file] --
   validate the integrity of a given file by read MD5 value from .md5 file\n";
7 }
```


解析

testMD5 函数是为了验证 MD5 哈希算法的实现是否正确而设计的。它通过对一系列预定义的测试字符串进行哈希运算,并将计算结果与预期的哈希值进行比较,从而检查 MD5 算法是否按照预期工作。该函数对每个测试用例输出测试结果,标明每项是通过还是失败,这对于调试和验证算法实现的准确性至关重要。

3.3.4 计算文件 MD5 computeMD5

```
1 void computeMD5(const string& filePath) {  
2     ifstream file(filePath, ifstream::binary);  
3     if (!file) {  
4         cerr << "Cannot open file: " << filePath << endl;  
5         return;  
6     }  
7  
8     stringstream buffer;  
9     buffer << file.rdbuf();  
10    string contents = buffer.str();  
11  
12    MD5 md5;  
13    string result = md5.computeMD5(contents);  
14    cout << "MD5(" << filePath << ") = " << result << endl;  
15 }
```

解析

computeMD5 函数允许用户为指定的文件计算 MD5 哈希值。该函数首先尝试打开用户指定的文件,然后读取文件内容,并使用 MD5 算法计算其哈希值。计算得到的 MD5 值将显示给用户,使用户能够获取文件的数字指纹。此功能在需要验证文件完整性或在数据备份过程中检查文件是否发生变化时非常有用。

3.3.5 手动验证 MD5 validateMD5Manual

```
1 void validateMD5Manual(const string& filePath) {  
2     ifstream file(filePath, ifstream::binary);  
3     if (!file) {  
4         cerr << "Cannot open file: " << filePath << endl;  
5         return;  
6     }  
7  
8     stringstream buffer;  
9     buffer << file.rdbuf();
```

```
10  string contents = buffer.str();
11
12  MD5 md5;
13  string computedMD5 = md5.computeMD5(contents);
14
15  string inputMD5;
16  cout << "Enter the MD5 hash to validate: ";
17  cin >> inputMD5;
18
19  if (computedMD5 == inputMD5) {
20      cout << "MD5 verification passed: " << computedMD5 << endl;
21  }
22  else {
23      cout << "MD5 verification failed: " << computedMD5 << " (computed) != "
24      << inputMD5 << " (expected)" << endl;
25  }
```

解析

validateMD5Manual 函数提供了一种手动验证文件 MD5 哈希值的方法。用户需要输入一个文件路径和一个他们期望的 MD5 哈希值，程序则计算该文件的实际 MD5 哈希并将其与用户输入的哈希值进行比较。这种验证方式用于确认文件从源到目的地传输或存储过程中未被篡改，是数据完整性验证的常见方法。

3.3.6 通过.md5 文件验证 MD5 validateMD5File

```
1  void validateMD5File(const string& filePath, const string& md5FilePath) {
2      ifstream file(filePath, ifstream::binary);
3      if (!file) {
4          cerr << "Cannot open file: " << filePath << endl;
5          return;
6      }
7
8      stringstream buffer;
9      buffer << file.rdbuf();
10     string contents = buffer.str();
11
12     MD5 md5;
13     string computedMD5 = md5.computeMD5(contents);
14
15     ifstream md5File(md5FilePath);
```

```
16  if (!md5File) {
17      cerr << "Cannot open MD5 file: " << md5FilePath << endl;
18      return;
19  }
20
21  string fileMD5;
22  getline(md5File, fileMD5);
23
24  if (computedMD5 == fileMD5) {
25      cout << "MD5 verification passed: " << computedMD5 << endl;
26  }
27  else {
28      cout << "MD5 verification failed: " << computedMD5 << " (computed) != "
29      << fileMD5 << " (expected)" << endl;
30  }
```

解析

validateMD5File 函数扩展了 MD5 验证功能，允许用户提供一个包含预计算 MD5 哈希值的 .md5 文件。程序将读取此文件中的哈希值，并计算目标文件的 MD5 哈希，然后比较这两个值是否相同。这样的自动化验证过程简化了与大量文件或数据集进行工作时的完整性校验，特别是在自动化脚本或批处理操作中非常实用。

3.3.7 总结

命令行界面 (CLI) 为 MD5 哈希算法的实用性和可访问性提供了极大的增强。通过设计精简而功能丰富的命令行参数处理，CLI 成功地将 MD5 算法的强大功能与用户友好的接口结合起来，使用户能够轻松地进行文件哈希运算，以及验证文件的完整性和安全性。

首先，CLI 通过一组清晰定义的命令行参数来支持多种操作模式，包括生成哈希、测试算法正确性以及验证哈希值。这种设计不仅提高了程序的灵活性，也使其可以适应各种使用场景，从日常文件校验到自动化脚本处理。例如，通过 -t 选项可以快速测试和验证算法的实现是否正确，而 -c 和 -v 选项则分别允许用户计算文件的 MD5 哈希值和手动验证哈希值，确保数据的一致性和完整性。

此外，CLI 在用户体验方面做了很好的考虑，提供了详尽的帮助信息和错误处理。通过 showHelp 函数，CLI 为用户提供了命令的详细说明和使用方法，帮助新用户快速掌握程序的使用。在出现输入错误或操作失败时，程序会输出相应的错误信息，并指导用户

如何正确操作，从而减少用户的困惑和潜在的操作错误。

3.4 程序编译

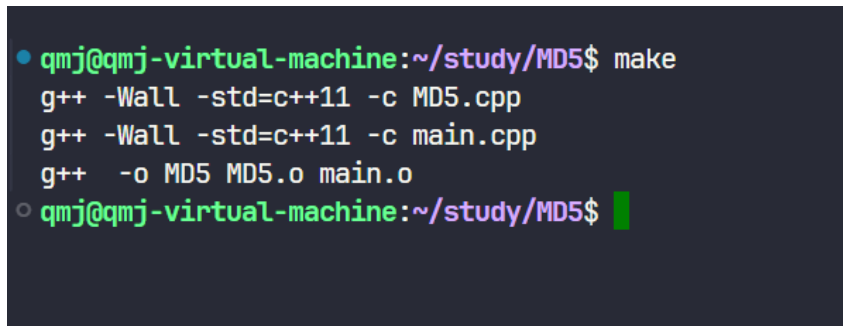
为了确保实验中的 MD5 程序可以在各种环境中一致地编译并运行，我们采用了 Makefile 来管理编译过程。Makefile 是一种被广泛使用的编译自动化工具，它定义了一组任务来自动化构建过程，简化了源代码到可执行文件的编译步骤。使用 Makefile 不仅可以减少重复的编译命令输入，而且还可以确保每次编译都是在相同的条件下进行，提高了构建的可靠性和效率。

编写如下 `Makefile`：

```
1  CXX = g++
2  CXXFLAGS = -Wall -std=c++11
3  LDFLAGS =
4
5  # 目标可执行文件名
6  TARGET = MD5
7
8  # 对象文件
9  OBJS = MD5.o main.o
10
11 # 默认目标
12 all: $(TARGET)
13
14 # 链接目标
15 $(TARGET): $(OBJS)
16     $(CXX) $(LDFLAGS) -o $@ $(OBJS)
17
18 # 编译源文件到对象文件
19 MD5.o: MD5.cpp MD5.h
20     $(CXX) $(CXXFLAGS) -c MD5.cpp
21
22 main.o: main.cpp MD5.h
23     $(CXX) $(CXXFLAGS) -c main.cpp
24
25 # 清理目标
26 clean:
27     rm -f $(OBJS) $(TARGET)
28
29 # 伪目标
30 .PHONY: all clean
```

其中，CXX 定义了使用的编译器为 g++，CXXFLAGS 设置了编译器的选项，启用所有警告并使用 C++11 标准进行编译，以确保代码的现代性和最佳实践。TARGET 和 OBJS 变量分别定义了最终生成的可执行文件名和需要编译的对象文件。Makefile 中的规则定义了如何编译源文件到目标文件，以及如何链接目标文件生成最终的可执行文件。

然后，我们可以在终端中执行 make 命令来编译 MD5 程序，如下所示：



```
• qmj@qmj-virtual-machine:~/study/MD5$ make
g++ -Wall -std=c++11 -c MD5.cpp
g++ -Wall -std=c++11 -c main.cpp
g++ -o MD5 MD5.o main.o
○ qmj@qmj-virtual-machine:~/study/MD5$
```

图 3.4.7: 执行 make 编译

程序将生成可执行文件和对象文件，如下所示：

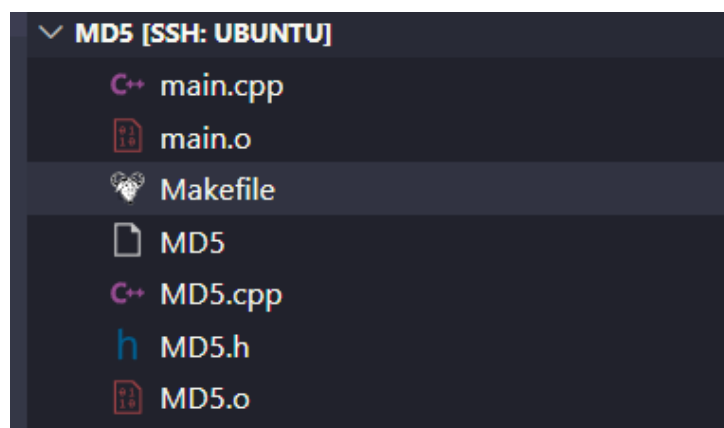
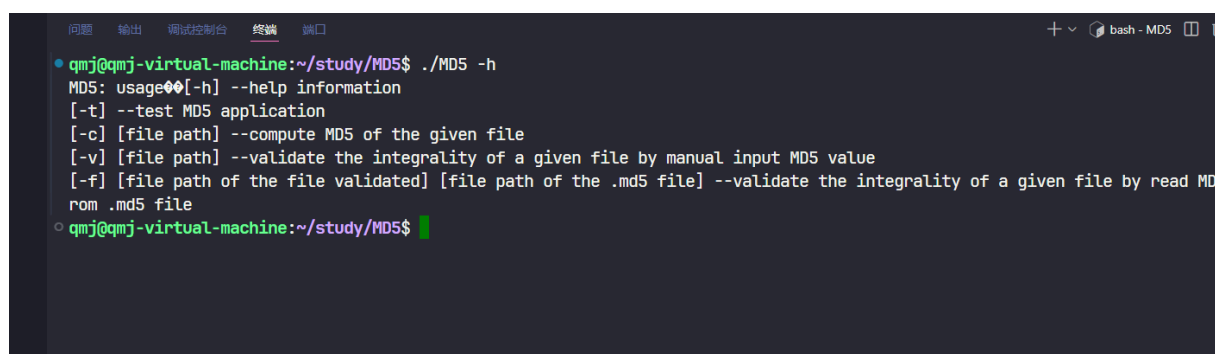


图 3.4.8: 生成的可执行文件

3.5 运行结果

根据上述的命令行选项和功能，我们可以通过不同的命令来执行 MD5 算法的测试、文件哈希计算和验证操作，分别如下所示：

3.5.1 显示帮助信息

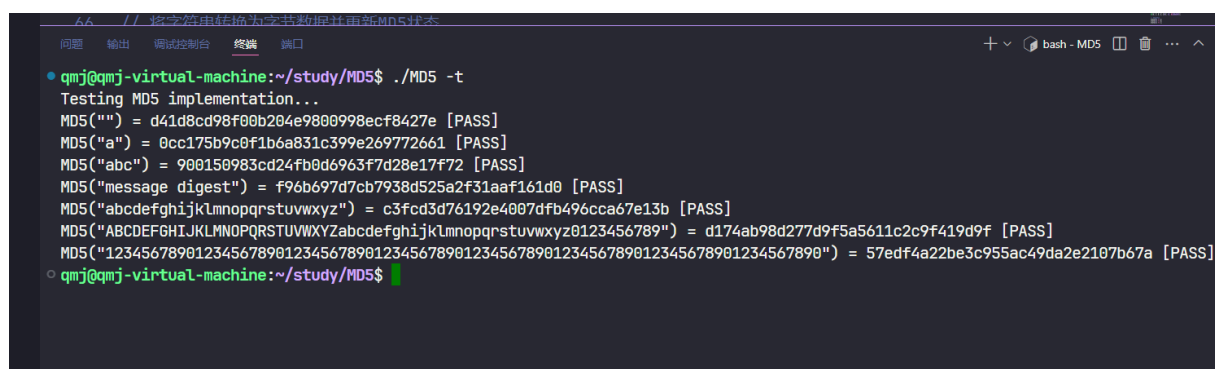


```
qmj@qmj-virtual-machine:~/study/MD5$ ./MD5 -h
MD5: usage [-h] --help information
[-t] --test MD5 application
[-c] [file path] --compute MD5 of the given file
[-v] [file path] --validate the integrity of a given file by manual input MD5 value
[-f] [file path of the file validated] [file path of the .md5 file] --validate the integrity of a given file by read MD5 file
qmj@qmj-virtual-machine:~/study/MD5$
```

图 3.5.9: 帮助信息

如图，通过 -h 选项可以显示程序的帮助信息，包括各个命令的功能和使用方法。这对于新用户来说非常有用，因为它提供了程序所有功能的快速概览和指引。

3.5.2 测试 MD5 算法



```
qmj@qmj-virtual-machine:~/study/MD5$ ./MD5 -t
Testing MD5 implementation...
MD5("") = d41d8cd98f00b204e9800998ecf8427e [PASS]
MD5("a") = 0cc175b9c0f1b6a831c399e269772661 [PASS]
MD5("abc") = 900150983cd24fb0d6963f7d28e17f72 [PASS]
MD5("message digest") = f96b697d7cb7938d525a2f31aaf161d0 [PASS]
MD5("abcdefghijklmnopqrstuvwxyz") = c3fcd3d76192e4007dfb496cca67e13b [PASS]
MD5("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789") = d174ab98d277d9f5a5611c2c9f419d9f [PASS]
MD5("1234567890123456789012345678901234567890123456789012345678901234567890") = 57edf4a22be3c955ac49da2e2187b67a [PASS]
qmj@qmj-virtual-machine:~/study/MD5$
```

图 3.5.10: 测试 MD5 算法

通过 -t 选项可以测试 MD5 算法的实现是否正确。图中显示全部正确，证明了 MD5 算法的正确性和可靠性。

3.5.3 计算文件的 MD5 哈希值



```
qmj@qmj-virtual-machine:~/study/MD5$ ./MD5 -c main.cpp
MD5(main.cpp) = 5f40748fa106548136157f81db71c50a
qmj@qmj-virtual-machine:~/study/MD5$
```

图 3.5.11: 计算文件 HASH

通过 -c 选项可以计算指定文件的 MD5 哈希值。这对于验证文件的完整性和安全性非常有用，用户可以通过哈希值来确认文件的唯一性。

3.5.4 手动验证文件的 MD5 哈希值

```
• qmj@qmj-virtual-machine:~/study/MD5$ ./MD5 -v main.cpp
Enter the MD5 hash to validate: aaaaaaaaaaaaaaaaaa
MD5 verification failed: 5f40748fa106548136157f81db71c50a (computed) != aaaaaaaaaaaaaaaaaa (expected)
• qmj@qmj-virtual-machine:~/study/MD5$ ./MD5 -v main.cpp
Enter the MD5 hash to validate: 5f40748fa106548136157f81db71c50a
MD5 verification passed: 5f40748fa106548136157f81db71c50a
• qmj@qmj-virtual-machine:~/study/MD5$
```

图 3.5.12: 输入 HASH 进行文件校验

通过 -v 选项可以手动验证文件的 MD5 哈希值。用户需要输入预期的哈希值，程序将计算文件的哈希值并与用户输入进行比较，以确保文件的完整性。这里分别展示了验证通过和验证失败的情况。

3.5.5 通过 .md5 文件验证文件的 MD5 哈希值

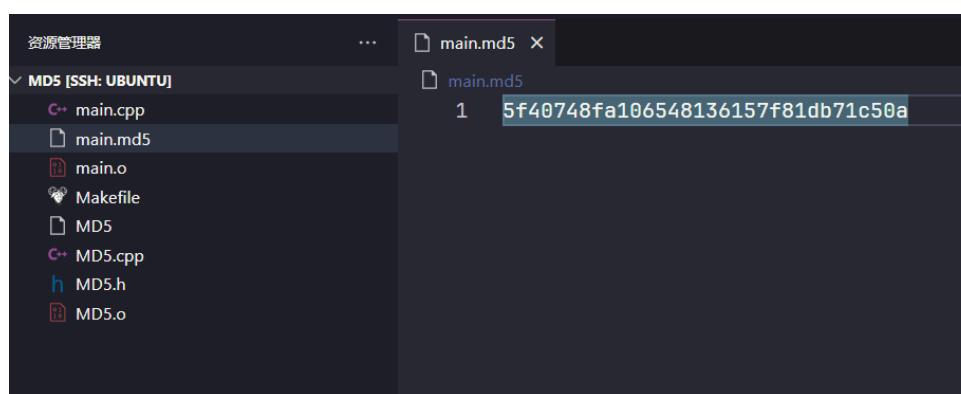


图 3.5.13: 创建.md5 文件

用户需要提供包含预期哈希值的 .md5 文件，这种验证方式适用于大量文件或数据集的批量处理。

```
• qmj@qmj-virtual-machine:~/study/MD5$ ./MD5 -f main.cpp main.md5
MD5 verification passed: 5f40748fa106548136157f81db71c50a
• qmj@qmj-virtual-machine:~/study/MD5$
```

图 3.5.14: 利用.md5 进行文件校验

然后进行文件校验，程序将读取 .md5 文件中的哈希值并与计算的哈希值进行比较，以确保文件的完整性。

4 实验遇到的问题及解决方法

在本次实验中，我遇到了一些问题，主要包括以下几个问题：

4.1 问题一：MD5 的大小端问题

MD5 算法在其规范中明确要求以小端模式处理字节数据。然而，不同的计算机架构可能采用不同的字节序。例如，x86 架构使用小端字节序，而一些旧的 PowerPC 或新的 ARM 架构（取决于配置）可能使用大端字节序。如果在一个使用大端字节序的系统上直接进行 MD5 计算，未经处理的数据将导致错误的哈希值。

解决方法

在本实验的 MD5 实现中，我通过 decode 和 encode 函数系统地处理了字节序问题，以确保无论在何种架构的系统上运行，都能产生一致的哈希值。这两个函数的作用是在算法处理数据之前和处理数据后正确地转换字节序。

decode 函数：该函数用于在读取原始数据到内部数据结构（如从字节流转换为 32 位整数数组）时正确处理字节序。它确保无论输入数据的原始字节序如何，内部处理总是基于统一的小端格式，符合 MD5 算法的规范要求。

```
1 void MD5::decode(const unsigned char input[], uint32_t output[], size_t  
len) {  
2     for (size_t i = 0, j = 0; j < len; i++, j += 4) {  
3         output[i] = ((uint32_t)input[j]) |  
4                     (((uint32_t)input[j + 1]) << 8) |  
5                     (((uint32_t)input[j + 2]) << 16) |  
6                     (((uint32_t)input[j + 3]) << 24);  
7     }  
8 }  
9
```

encode 函数：在最终生成的 32 位整数哈希值需要转换回字节序列以输出十六进制字符串时，encode 函数确保无论目标平台是大端还是小端，输出的哈希值的表示都是正确的。

```
1 void MD5::encode(uint32_t input[], unsigned char output[], size_t len)  
{  
2     for (size_t i = 0, j = 0; j < len; i++, j += 4) {  
3         output[j] = input[i] & 0xff;  
4         output[j + 1] = (input[i] >> 8) & 0xff;  
5         output[j + 2] = (input[i] >> 16) & 0xff;  
6         output[j + 3] = (input[i] >> 24) & 0xff;  
7     }  
8 }  
9
```

通过这种方式，decode 和 encode 函数不仅解决了跨平台的字节序问题，还增强了算法的通用性和一致性，确保了无论在何种系统上，MD5 算法都能正确地计算出数据的哈希值。这是实现加密算法时考虑系统兼容性和数据一致性的一个典型例子。

4.2 问题二：命令程序的参数

在本实验中，我通过使用 C++ 标准库中的 `argc` 和 `argv` 参数来获取和处理命令行输入。这两个参数由主函数 `int main(int argc, char* argv[])` 接收，其中 `argc` 表示命令行参数的数量，`argv` 是一个指针数组，指向各个参数的字符串表示。

解决方法

- **参数计数 (`argc`)** : `argc` 的值反映了传递给程序的命令行参数的总数，包括程序本身的名称。通过检查 `argc` 的值，可以判断用户是否提供了足够的参数来执行请求的操作。例如，如果 `argc` 小于预期的参数数量，程序将显示帮助信息或错误消息。
- **参数值 (`argv`)** : `argv` 数组包含了每个参数的具体值，`argv[0]` 是程序的名称，`argv[1]` 是传递给程序的第一个参数，依此类推。这些参数可以是文件路径、操作选项（如 `-h` 或 `-t`）或其他需要的数据。
- **参数解析** : 在获取了参数数量和值之后，我使用简单的条件语句来解析这些参数，并根据不同的参数执行相应的功能。例如，通过比较 `argv` 中的字符串来确定是否需要显示帮助信息、执行测试或处理文件。这种方法虽然基础，但对于简单的命令行程序来说足够有效。

```
1 if (strcmp(argv[1], "-h") == 0) {  
2     showHelp();  
3 } else if (strcmp(argv[1], "-t") == 0) {  
4     testMD5();  
5 } else if (strcmp(argv[1], "-c") == 0 && argc == 3) {  
6     computeMD5(argv[2]);  
7 }
```

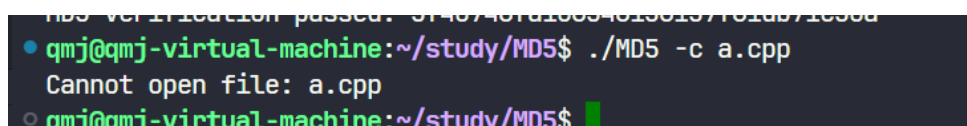
4.3 问题三：文件读取的异常处理

文件读取过程中可能遇到各种问题，例如文件不存在、权限不足或文件损坏等。

解决方法

检查文件存在性：在尝试打开文件之前，首先检查文件是否存在。这可以通过标准库函数如 `std::ifstream` 的状态检查来实现。如果文件不存在，程序会立即返回错误消息，不再继续执行。

```
1 std::ifstream file(filePath, std::ifstream::binary);  
2 if (!file) {  
3     std::cerr << "Cannot open file: " << filePath << std::endl;  
4     return;  
5 }
```



```
md5 verification passed: 3f40740fd100540100197f01ab71c30d
• qmj@qmj-virtual-machine:~/study/MD5$ ./MD5 -c a.cpp
Cannot open file: a.cpp
• qmj@qmj-virtual-machine:~/study/MD5$
```

图 4.3.15: 找不到文件处理

5 实验结论

本次实验通过实现和测试 MD5 哈希算法的命令行应用程序，成功地展示了如何在实际编程中处理数据安全性和程序稳定性的关键问题。通过对 MD5 算法的详细实现和优化，以及针对程序输入输出、异常处理和跨平台兼容性的深入分析，实验不仅提高了对加密哈希技术的理解，还强化了在实际软件开发中应对复杂问题的能力。总结来说，这次实验不仅成功实现了一个功能完整的 MD5 命令行工具，还通过面对和解决实际编程中的挑战，加深了对理论知识的理解和应用。这些经验将对未来在更广泛的编程和技术问题中的应对提供极大的帮助和指导。