

软件工程 实验报告

****

**磁盘文件管理工具**

**学 生：**  孙浩浩

**学 号： 201706060328**

**班 级： 网络172**

**系 别： 计算机系**

**学 院： 电子信息与人工智能学院**

**一、可行性及必要性分析**

由需求调研所得数据，只要是经常使用电脑的人，或多或少都会遇到由磁盘内存不足导致的电脑卡顿情况，而且大部分人是通过清理软件来进行清理磁盘的，但有些清理软件是通过将磁盘的内容强行写入硬盘之中，导致磁盘寿命被降低，从而导致电脑使用寿命被降低。

但是我们这个项目所采用的是借助电脑自带的删除功能进行删除磁盘的重复文件，不会对电脑造成伤害，也比电脑自带的清理工具更加容易使用。

**二、需求性分析**

（一）功能性需求:

1. 根据文件名删除重复文件
2. 根据MD5值删除重复文件
3. 根据给定的字符串删除重复文件
4. 删除所有重复文件
5. 输入输出要求：

输入：用户输入相应功能编号

输出：功能提示，重复的文件名和路径，已删除的文件名

1. 数据管理能力需求：

数据采集输入源：手工键盘输入

数据采集的处理：根据输入的编号调用相应的功能

1. 故障处理需求

正常使用时不应出错。

**三、系统设计**

1．系统设计目标。

1）可以扫描文件

2）可以打印文件

3）可以删除文件

2.总体设计。

1）扫描文件

2）算法文件

3）文件管理文件

4）测试工具文件

5）测试文件

3.详细设计。

1）扫描文件

①扫描目录

②删除文件

1. 算法文件

①位运算函数

②循环左移

③初始化

④重新设值

⑤计算一块的MD5值

⑥填充操作

⑦整型数据转成16进制字符串

⑧得到一个字符串的MD5值

⑨得到一个文件的MD5值

1. 文件管理文件

①根据MD5值得到相应的文件

②扫描目录

③得到重复文件表

④根据文件名删除文件

⑤根据MD5值删除文件

⑥删除所有重复文件只保留一份文件

⑦根据‘关键字‘删除文件

⑧打印重复文件表

⑨打印所有文件

1. 测试工具文件

①扫描文件

②展示重复文件

③根据文件名删除文件

④根据MD5值删除文件

⑤删除含有‘关键字’的重复文件

⑥删除所有重复文件只保留一份

1. 测试文件

①测试整型数据转成16进制字符串

②测试得到一个字符串的MD5值

③测试得到一个文件的MD5值

④测试扫描目录

⑤测试文件管理文件

⑥测试根据MD5值删除文件

⑦根据文件名删除文件

⑧测试测试工具

**四、编码**

1)算法模块：

①算法类：

#pragma once

#include<string>

#define Chunk\_Byte 64

typedef unsigned int uint;

class MD5

{

public:

//位运算的函数,F,G,H,I

/\*

F(B,C,D) = (B & C) | ((~B) & D)

F(x,y,z) = (x & y) | ((~x) & z)

G(B,C,D) = (B & D) | ( C & (~D))

G(x,y,z) = (x & z) | ( y & (~z))

H(B,C,D) = B ^ C ^ D

H(x,y,z) = x ^ y ^ z

I(B,C,D) = C ^ (B | (~D))

I(x,y,z) = y ^ (x | (~z))

\*/

uint F(uint B, uint C, uint D)

{

return (B&C) | ((~B)&D);

}

uint G(uint B, uint C, uint D)

{

return (B&D) | (C&(~D));

}

uint H(uint B, uint C, uint D)

{

return B^C^D;

}

uint I(uint B, uint C, uint D)

{

return C ^ (B | (~D));

}

//循环左移

uint LeftShift(uint number, int shift\_number)

{

/\*

number 1byte --->1100 1010

number 左移一位 1001 0101

(number<<1)|(number>>7) --->

number<<1: 1001 0100

number>>7: 0000 0001

或运算

1001 0101

\*/

return (number << shift\_number) |

(number >> (32 - shift\_number));

}

MD5();

//初始化

void init();

//重新设值

void reset();

//计算一个chunk的MD5值

void md5(uint\* chunk);

//填充操作,进行填充，再计算MD5

void fill\_md5();

//把一个整数数据转成对应的16进制字符串

std::string change(uint n);

std::string get\_Sting\_md5(const std::string& str);

std::string get\_File\_md5(const char\* filePath);

private:

//循环移位的位数

static int Left\_Shift[64];

//k[i]= (size\_t)(abs(sin(i + 1)) \* pow(2, 32));

uint k[64];

//数据块：64字节

char chunk[Chunk\_Byte];

//填充相关变量:

// 最后一块数据的字节

uint Last\_Byte;

// 总字节数

uint Sum\_Byte;

//MD5信息的A,B,C,D

uint A;

uint B;

uint C;

uint D;

};

②类成员函数具体实现

#include "md5.h"

#include<math.h>

#include<iostream>

#include<fstream>

//static成员初始化

int MD5::Left\_Shift[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 };

MD5::MD5()

{

init();

}

void MD5::init()

{

//初始化k[i]

//K[i] = floor(2^(32) \* abs(sin(i + 1)))因为此处i从0开始，所以需要sin(i + 1) floor是取整的意思

for (int i = 0;i < 64;++i)

{

k[i]=static\_cast<uint>(abs(sin(i + 1.0))\*pow(2.0, 32));

}

reset();

}

//重新设值

void MD5::reset()

{

//初始化A,B,C,D

A = 0x67452301;

B = 0xefcdab89;

C = 0x98badcfe;

D = 0x10325476;

//初始化chunk

memset(chunk, 0, Chunk\_Byte);

Last\_Byte = Sum\_Byte = 0;

}

//计算一个chunk的MD5值

void MD5::md5(uint\* chunk)

{

int a = A;

int b = B;

int c = C;

int d = D;

int f, g;

//4byte是一个处理单元，共执行64次

for (int i = 0;i < 64;++i)

{

//位运算，F,G,H,I

/\*

if (0 <= i < 16) g = i;

if (16 <= i < 32) g = (5 \* i + 1) % 16;

if (32 <= i < 48) g = (3 \* i + 5) % 16;

if(48 <= i < 63) g = (7 \* i) % 16;

\*/

//0~15:F

if (0 <= i && i <= 15)

{

f = F(b, c, d);

g = i;

}

//16~31:G

else if (16 <= i && i <= 31)

{

f = G(b, c, d);

g = (5 \* i + 1) % 16;

}

//32~47:H

else if (32 <= i && i <= 47)

{

f = H(b, c, d);

g = (3 \* i + 5) % 16;

}

//48~63:I

else

{

f = I(b, c, d);

g = (7 \* i) % 16;

}

//更新a,b,c,d，加法+循环左移

int ta = a;

int tb = b;

int tc = c;

int td = d;

a = td;

b = b + LeftShift(a + f + chunk[g] + k[i], Left\_Shift[i]);

c = tb;

d = tc;

}

//更新A,B,C,D

A += a;

B += b;

C += c;

D += d;

}

//填充操作

void MD5::fill\_md5()

{

//填充冗余信息：第一位填1，其余填0

//任何情况下，都至少填充一个字节的冗余信息

//获取第一个要填充的位置

char \*p = chunk + Last\_Byte;

//首先填充1 byte 的冗余信息：1000 0000

//0x80(16进制)==128(10进制)

\*p++= 0x80;

//剩余字节的位置

int Surplus\_Byte = Chunk\_Byte - Last\_Byte - 1;

//如果剩余字节不够8字节，不能填充长度信息

//应该先处理这块数据，然后再去构建一个新的数据块，前440位填0，最后的64位填长度xinx

if (Surplus\_Byte < 8)

{

//剩余位全部填0

memset(p, 0, Surplus\_Byte);

md5((uint\*)chunk);

//构建一个新的chunk

memset(chunk, 0, Chunk\_Byte);

}

else

{

//剩余位全部补0

memset(p, 0, Surplus\_Byte);

}

//最终在最后一块数据的最后64位填充原始信息

unsigned long long Sum\_Bits = Sum\_Byte;

Sum\_Bits \*= 8;

((unsigned long long \*)chunk)[7] = Sum\_Bits;

md5((uint\*)chunk);

}

std::string MD5::change(uint n)

{

static std::string arr = "0123456789abcdef";

std::string ret;

//获取每个字节数据

for (int i = 0;i < 4;++i)

{

//获取一个字节的数据

/\*

将12345678的高位通过右移8位，然后通过和0xff(1111 1111)相与

\*/

int cur = n >> (i \* 8) & 0xff;

//数据转成16进制字符

std::string cur\_str;

//÷16获取高位，%16获取低位 在字节内部是没有逆序的

cur\_str += arr[cur / 16];

cur\_str += arr[cur % 16];

//在字节之间是逆序的

ret += cur\_str;

}

return ret;

}

std::string MD5::get\_Sting\_md5(const std::string& str)

{

if (str.empty())

{

return change(A).append(change(B)).append(change(C)).append(change(D));

}

Sum\_Byte = str.size();

uint chunk\_Num = Sum\_Byte / Chunk\_Byte;

const char\* str\_Ptr = str.c\_str();

for (int i = 0;i < (int)chunk\_Num;++i)

{

memcpy(chunk, str\_Ptr + i \* Chunk\_Byte, Chunk\_Byte);

md5((uint\*)chunk);

}

//计算最后一个数据：需要填充

Last\_Byte = Sum\_Byte % Chunk\_Byte;

memcpy(chunk, str\_Ptr + chunk\_Num \* Chunk\_Byte, Last\_Byte);

fill\_md5();

return change(A).append(change(B)).append(change(C)).append(change(D));

}

/\*seekg 在输入序列中设置位置

istream& seekg (streampos pos);pos你想获取位置的指针

istream& seekg (streamoff off, ios\_base::seekdir way);

off 起始的偏移量，可正可负

way 基地址只能是：

ios：：beg：表示输入流的开始位置

ios：：cur：表示输入流的当前位置

ios：：end：表示输入流的结束位置

tellg获取输入序列中的位置

streampos tellg();

gcount 获取字符计数

返回最后一次对该对象执行的无格式输入操作提取的字符数。

修改此函数返回值的非格式化输入操作有:get、getline、ignore、peek、read、readsome、putback和unget。

但是请注意，peek、putback和unget实际上并没有提取任何字符，因此gcount在调用任何字符后总是返回0。

\*/

std::string MD5::get\_File\_md5(const char\* filePath)

{

std::ifstream fin(filePath, std::ifstream::binary);

//std::fstream fin(filePath, std::ifstream::binary | std::ifstream::app);

//std::fstream fin(filePath);

if (!fin.is\_open())

{

std::cout << filePath;

perror("file open failed!!!");

return "";

}

while (!fin.eof())

{

fin.read(chunk, Chunk\_Byte);

//是否读取到64个字节的内容

if (Chunk\_Byte != fin.gcount())

{

//说明是最后一块数据

break;

}

Sum\_Byte += Chunk\_Byte;

md5((uint\*)chunk);

}

Last\_Byte = fin.gcount();

//gcount:可以调用多次，始终返回最近一次读入的字节数

Sum\_Byte += Last\_Byte;

fill\_md5();

return change(A).append(change(B)).append(change(C)).append(change(D));

}

1. 扫描文件模块

①扫描文件函数

#pragma once

#include<io.h>

#include<string>

#include<vector>

#include<iostream>

#include<unordered\_set>

void search\_dir(const std::string& path, std::unordered\_set<std::string>& sum\_file);

void delete\_file(const char\* filename);

②扫描文件函数的具体实现

#include"file.h"

void search\_dir(const std::string& path, std::unordered\_set<std::string>& sum\_file)

{

std::string match = path + "\\" + "\*.\*";

//std::string match = path;

\_finddata\_t file\_attr;

long handle = \_findfirst(match.c\_str(), &file\_attr);

if (handle == -1)

{

perror("search failed!!!");

std::cout << match << std::endl;

return;

}

do

{

if (file\_attr.attrib & \_A\_SUBDIR)

{

if (strcmp(file\_attr.name, ".") != 0 && strcmp(file\_attr.name, "..") != 0)

{

//当前为目录，继续搜索

search\_dir(path + "\\" + file\_attr.name, sum\_file);

}

}

else

{

//当前为文件，则保存

sum\_file.insert(path + "\\" + file\_attr.name);

}

} while (\_findnext(handle, &file\_attr) == 0);

\_findclose(handle);

}

void delete\_file(const char\* filename)

{

if (remove(filename) == 0)

{

std::cout << "delete file:" << filename << "success!!!" << std::endl;

}

else

{

perror("delete file failed!!!");

}

}

1. 文件管理模块

①文件管理类

#pragma once

#include<string>

#include<unordered\_set>

#include<unordered\_map>

#include"md5.h"

#include<utility>

class file\_manager

{

public:

void get\_md5\_to\_file();

void scanner\_dir(const std::string& path);

void get\_copy\_list();

//所有的删除，保证一个文件不存在副本

void delete\_by\_name(const std::string& name);

void delete\_by\_md5(const std::string& md5);

void delete\_all\_copy();

//删除所有包含'match'的文件

void delete\_by\_match(const std::string& match);

void show\_copy\_list();

void show\_all\_file();

private:

std::unordered\_set<std::string> \_file;

//md5映射文件

std::unordered\_multimap<std::string, std::string> \_md5\_to\_file;

//文件映射md5

std::unordered\_map<std::string, std::string> \_file\_to\_md5;

MD5 \_md5;

};

②文件管理类的具体实现

#include"file\_manager.h"

#include"file.h"

#include<utility>

void file\_manager::scanner\_dir(const std::string& path)

{

//清理容器

\_file.clear();

search\_dir(path, \_file);

//std::cout << "all file" << std::endl;

show\_all\_file();

get\_md5\_to\_file();

show\_copy\_list();

std::cout << "Only duplicate files" << std::endl;

get\_copy\_list();

//std::cout << "copy list" << std::endl;

show\_copy\_list();

show\_all\_file();

}

void file\_manager::get\_md5\_to\_file()

{

\_md5\_to\_file.clear();

for (const auto &a : \_file)

{

//计算新文件前重置

\_md5.reset();

//string 转换成 char \*

//如果要将string直接转换成const char \*类型。string有2个函数可以运用。

//一个是.c\_str()，一个是data成员函数。

\_md5\_to\_file.insert(make\_pair(\_md5.get\_File\_md5(a.c\_str()), a));

}

}

//只保留内容重复文件的结果

void file\_manager::get\_copy\_list()

{

\_file\_to\_md5.clear();

//不要用范围for,涉及删除操作

auto it = \_md5\_to\_file.begin();

while (it != \_md5\_to\_file.end())

{

//查找每个MD5对应所有文件的结果

if (\_md5\_to\_file.count(it->first) > 1)

{

//equal\_range 返回值：pair<beginIt,endIt>:[beginIt,endIt)--->迭代器遍历时连续

auto pair\_it = \_md5\_to\_file.equal\_range(it->first);

auto begin = pair\_it.first;

while (begin != pair\_it.second)

{

//只存放重复文件的映射关系

\_file\_to\_md5.insert(make\_pair(begin->second, begin->first));

++begin;

}

//下一个不同的MD5的起始位置

it = pair\_it.second;

}

else

{

//先删除，再更新

\_file.erase(it->second);

//erase的返回值，被删除元素的下一个位置

it = \_md5\_to\_file.erase(it);

}

}

}

//所有的删除，保证一个文件不存在副本

void file\_manager::delete\_by\_name(const std::string& name)

{

if (\_file\_to\_md5.count(name) == 0)

{

std::cout << name << "\tnot exist!!!" << std::endl;

return;

}

std::string cur\_md5 = \_file\_to\_md5[name];

std::cout << name << "--->" << \_md5\_to\_file.count(cur\_md5) << std::endl;

auto pair\_it = \_md5\_to\_file.equal\_range(cur\_md5);

auto cur\_it = pair\_it.first;

int count = 0;

while (cur\_it != pair\_it.second)

{

if (cur\_it->second != name)

{

\_file.erase(cur\_it->second);

\_file\_to\_md5.erase(cur\_it->second);

delete\_file(cur\_it->second.c\_str());

++count;

}

++cur\_it;

}

cur\_it = pair\_it.first;

while (cur\_it != pair\_it.second)

{

if (cur\_it->second != name)

{

//key==md5

\_md5\_to\_file.erase(cur\_it);

pair\_it = \_md5\_to\_file.equal\_range(cur\_md5);

cur\_it = pair\_it.first;

}

++cur\_it;

}

std::cout << "delete files ：" << count << std::endl;

}

void file\_manager::delete\_by\_md5(const std::string& md5)

{

//md5映射file

if (\_md5\_to\_file.count(md5) == 0)

{

std::cout << md5 << "not exist!!!" << std::endl;

return;

}

//删除时需要保留一份，规定保留第一个文件

auto pair\_it = \_md5\_to\_file.equal\_range(md5);

std::cout << "md5值：" << md5 << "---have---" << \_md5\_to\_file.count(md5) << std::endl;

auto cur\_it = pair\_it.first;

++cur\_it;

int count = 0;

while (cur\_it != pair\_it.second)

{

\_file.erase(cur\_it->second);

\_file\_to\_md5.erase(cur\_it->second);

//文件从磁盘中删除

delete\_file(cur\_it->second.c\_str());

++count;

++cur\_it;

}

//更新一下MD5到file的映射

cur\_it = pair\_it.first;

++cur\_it;

//erase(first,last)----> 删除区间[first,last)

\_md5\_to\_file.erase(cur\_it, pair\_it.second);

std::cout << "delete files number：" << count << std::endl;

}

//所有重复的文件只保留一个

void file\_manager::delete\_all\_copy()

{

std::unordered\_set<std::string> md5\_set;

//找出所有的md5值

for (const auto& a : \_md5\_to\_file)

{

md5\_set.insert(a.first);

}

for (const auto& b : md5\_set)

{

delete\_by\_md5(b);

}

}

//删除所有包含'match'的文件

void file\_manager::delete\_by\_match(const std::string& match)

{

std::unordered\_set<std::string> all\_file;

//遍历所有文件

for (const auto& a : \_file)

{

if (a.find(match) != std::string::npos)

all\_file.insert(a);

}

for (const auto & b : all\_file)

{

if (\_file\_to\_md5.count(b) != 0)

delete\_by\_name(b);

}

}

void file\_manager::show\_copy\_list()

{

std::cout << "+++++++++++++++++++++++" << std::endl;

std::cout << "show\_copy\_list" << std::endl;

auto it = \_md5\_to\_file.begin();

int total = \_md5\_to\_file.size();

int count = 0;

while (it != \_md5\_to\_file.end())

{

//md5值相同的显示在一起

int idx = 1;

auto pair\_it = \_md5\_to\_file.equal\_range(it->first);

auto cur\_it = pair\_it.first;

std::cout << "cut md5:" << it->first << std::endl;

while (cur\_it != pair\_it.second)

{

std::cout << "\t第" << idx << "文件" << "\t";

std::cout << cur\_it->second << std::endl;

++count;

++cur\_it;

++idx;

}

it = pair\_it.second;

}

std::cout << "文件总数:" << total << "\t" << count << std::endl;

std::cout << "+++++++++++++++++++++++" << std::endl;

}

void file\_manager::show\_all\_file()

{

std::cout << "=======================" << std::endl;

std::cout << "show\_all\_file" << std::endl;

for (const auto& a : \_file)

{

std::cout << a << std::endl;

}

std::cout << "file count:" << \_file.size() << std::endl;

std::cout << "=======================" << std::endl;

}

1. 测试工具模块

①测试工具类

#pragma once

#include"file\_manager.h"

#include<iostream>

using std::cout;

using std::cin;

using std::endl;

class file\_manager\_tool

{

public:

void scanner();

void delete\_by\_md5();

void delete\_by\_name();

void delete\_all\_copy();

void delete\_by\_match();

void show\_all\_file();

void show\_copy();

private:

file\_manager \_fm;

};

②测试工具类的具体实现

#include"file\_manager\_tool.h"

void file\_manager\_tool::scanner()

{

cout << "请输入要扫描的文件夹：" << endl;

std::string path;

getline(cin, path);

\_fm.scanner\_dir(path);

}

void file\_manager\_tool::delete\_by\_md5()

{

cout << "请输入要删除的文件夹的MD5值：" << endl;

std::string md5;

cin >> md5;

\_fm.delete\_by\_md5(md5);

}

void file\_manager\_tool::delete\_by\_name()

{

cout << "请输入要删除的文件名：" << endl;

std::string file\_name;

getline(cin, file\_name);

\_fm.delete\_by\_name(file\_name);

}

void file\_manager\_tool::delete\_all\_copy()

{

\_fm.delete\_all\_copy();

}

void file\_manager\_tool::delete\_by\_match()

{

cout << "请输入要删除文件所包含的内容：" << endl;

std::string md5;

cin >> md5;

\_fm.delete\_by\_match(md5);

}

void file\_manager\_tool::show\_all\_file()

{

\_fm.show\_all\_file();

}

void file\_manager\_tool::show\_copy()

{

\_fm.show\_copy\_list();

}

1. 测试模块

#include"md5.h"

#include<iostream>

#include<stdlib.h>

#include"file.h"

#include"file\_manager.h"

#include"file\_manager\_tool.h"

using std::endl;

using std::cout;

void test\_change()

{

MD5 m;

uint n = 0x12345678;

cout << m.change(n) << endl;

/\*std::string s = m.change(n);

for (auto a : s)

{

cout << a;

++a;

}

cout << endl;\*/

}

void test\_get\_Sting\_md5()

{

MD5 m;

std::string s = "123456789abcdefgh";

cout << m.get\_Sting\_md5(s) << endl;

std::string ss = "123456789abcdefgi";

cout << m.get\_Sting\_md5(ss) << endl;

}

void test\_get\_File\_md5()

{

MD5 m;

const char\* f1 = "test\_a.txt";

const char\* f2 = "test\_b.txt";

cout << m.get\_File\_md5(f1) << endl;

//需要重置一下

m.reset();

cout << m.get\_File\_md5(f2) << endl;

}

void test\_search()

{

std::string path;

cout << "please input search dir:" << endl;

getline(std::cin, path);

std::unordered\_set<std::string> sum\_file;

search\_dir(path, sum\_file);

for (const auto & a : sum\_file)

{

cout << a << endl;

}

}

void test\_file\_manage()

{

file\_manager fm;

std::string path = "E:\\去重行动测试文件";

//std::string path = "E:\\去重行动测试文件 - 副本";

fm.scanner\_dir(path);

}

void test\_delete\_by\_md5()

{

file\_manager fm;

std::string path = "E:\\去重行动测试文件";

fm.scanner\_dir(path);

fm.delete\_by\_md5("38791e3e96967d228dce7a359e870611");

//fm.scanner\_dir(path);

}

void test\_delete\_by\_name()

{

file\_manager fm;

std::string path = "E:\\去重行动测试文件";

fm.scanner\_dir(path);

fm.delete\_by\_name("20.doc");

}

void menu()

{

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "1:扫描文件 2:展示重复文件 3:根据名字删除文件" << endl;

cout << "4:根据MD5删除文件 5:删除含有'matched'的重复文件 6:所有重复文件均删除" << endl;

cout << "0:退出" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

}

void test\_tool()

{

file\_manager\_tool fmt;

int input;

do

{

menu();

std::string rubbish;

cout << "请输入选项：";

cin >> input;

getline(cin, rubbish);

switch (input)

{

case 1:

fmt.scanner();

break;

case 2:

fmt.show\_copy();

break;

case 3:

fmt.delete\_by\_name();

break;

case 4:

fmt.delete\_by\_md5();

break;

case 5:

fmt.delete\_by\_match();

break;

case 6:

fmt.delete\_all\_copy();

break;

case 0:

break;

default:

break;

}

} while (input != 0);

}

int main()

{

//test\_change();

//test\_get\_Sting\_md5();

//test\_get\_File\_md5();

//test\_search();

//test\_file\_manage();

//test\_delete\_by\_md5();

//test\_delete\_by\_name();

test\_tool();

system("pause");

return 0;

}

**五、系统测试**

void test\_change()

{

MD5 m;

uint n = 0x12345678;

cout << m.change(n) << endl;

/\*std::string s = m.change(n);

for (auto a : s)

{

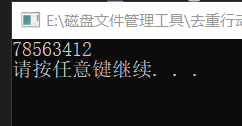
cout << a;

++a;

}

cout << endl;\*/

}



成功将0x12345678输出为78563412，可见到达了预期目标，也实现了字节内顺序、字节间逆序。

void test\_get\_Sting\_md5()

{

MD5 m;

std::string s = "123456789abcdefgh";

cout << m.get\_Sting\_md5(s) << endl;

std::string ss = "123456789abcdefgi";

cout << m.get\_Sting\_md5(ss) << endl;

}



输入的两个字符串只有一个字符不相同，但输出计算他们的MD5是完全不同，可见实现了雪崩效应，只改变一个字符，所计算得到的MD5值就完全不同

void test\_get\_File\_md5()

{

MD5 m;

const char\* f1 = "test\_a.txt";

const char\* f2 = "test\_b.txt";

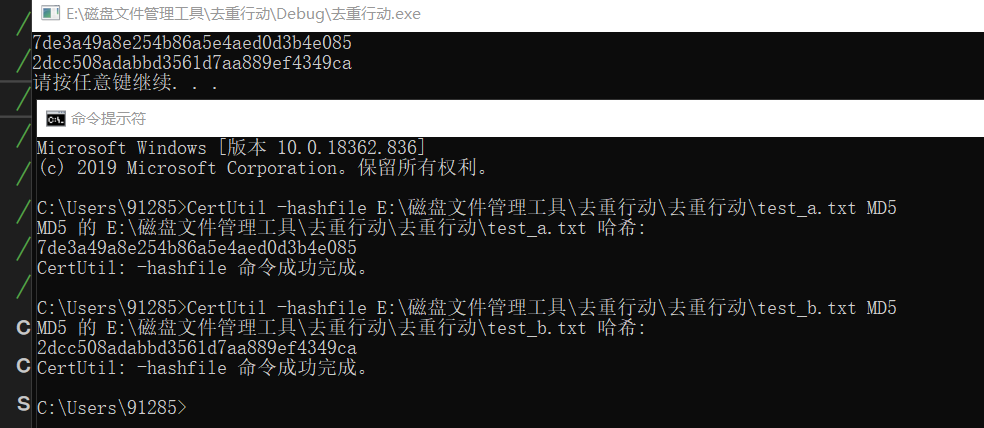
cout << m.get\_File\_md5(f1) << endl;

//需要重置一下

m.reset();

cout << m.get\_File\_md5(f2) << endl;

}



使用我们自己编写的MD5算法和系统自带的MD5算法计算同一文件，所得到的MD5值是完全相同的，可见我们自己编写的MD5算法和系统自带的MD5算法的效果是一样的，也表示我们的算法位后续的代码提供可靠的MD5值。

void test\_search()

{

std::string path;

cout << "please input search dir:" << endl;

getline(std::cin, path);

std::unordered\_set<std::string> sum\_file;

search\_dir(path, sum\_file);

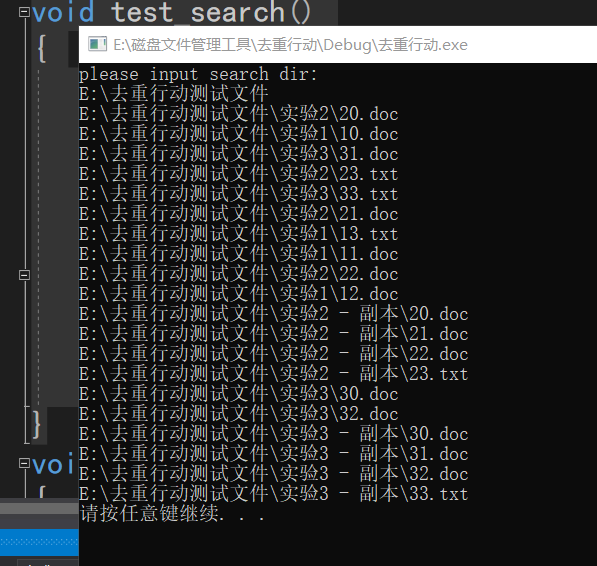
for (const auto & a : sum\_file)

{

cout << a << endl;

}

}



可见我们自己编写的扫描文件，是可以将某一文件夹下的所有文件扫描并打印出来且得到的结果就是想要的结果没有一丝差别。

void test\_file\_manage()

{

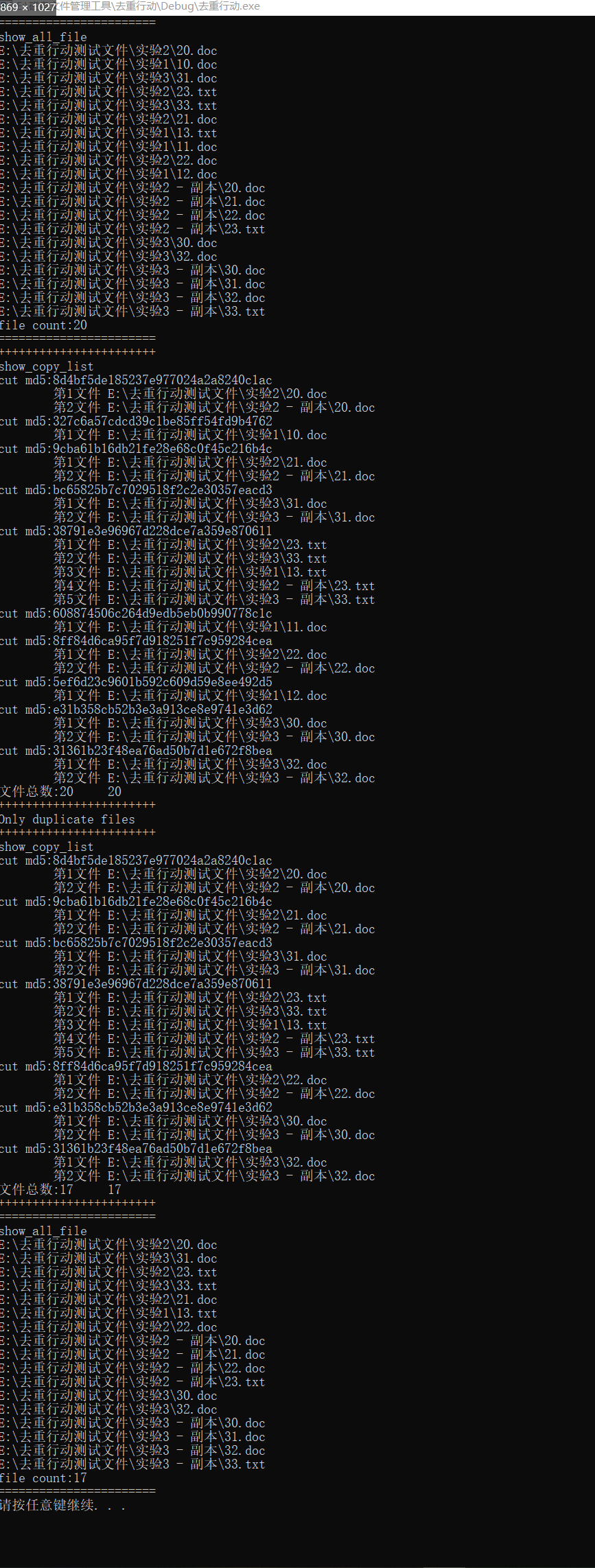
file\_manager fm;

std::string path = "E:\\去重行动测试文件";

//std::string path = "E:\\去重行动测试文件 - 副本";

fm.scanner\_dir(path);

}



成功将文件总数打印出来且将重复文件总数也一并打印出来，给用户提供了更加可观的界面，且可以让用户轻易地得到他们想要的东西，增加了用户体验感。

void test\_delete\_by\_md5()

{

file\_manager fm;

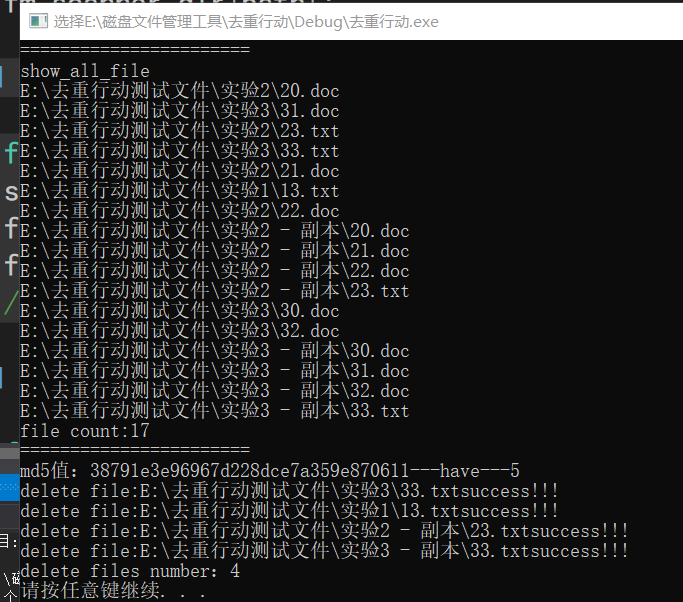
std::string path = "E:\\去重行动测试文件";

fm.scanner\_dir(path);

fm.delete\_by\_md5("38791e3e96967d228dce7a359e870611");

//fm.scanner\_dir(path);

}



上面和test\_file\_manage()一样

成功地实现根据用户输入的文件MD5值删除该MD5值对应文件的相应重复文件，并没有删除其他文件，用户可以放心使用该功能。

void test\_delete\_by\_name()

{

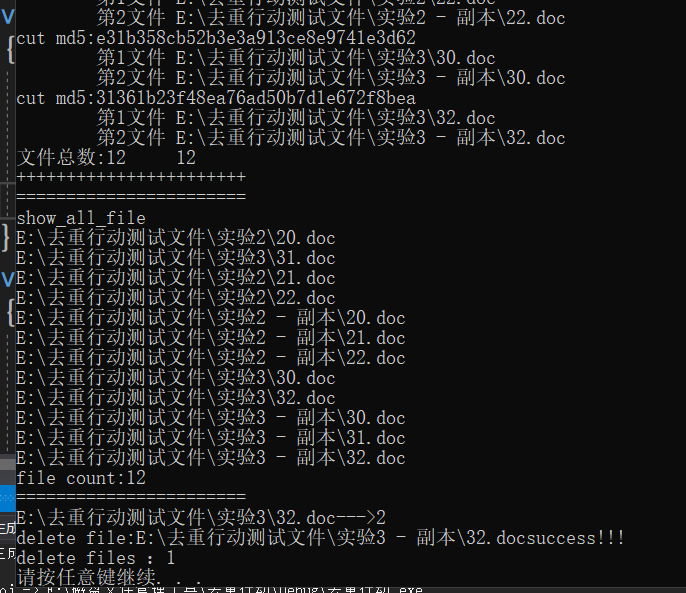
file\_manager fm;

std::string path = "E:\\去重行动测试文件";

fm.scanner\_dir(path);

fm.delete\_by\_name("20.doc");

}



上面和test\_file\_manage()一样

成功地实现了根据用户输入的文件名，计算其相对应的MD5值，在根据该MD5值找到对应的重复文件，从而进行删除，并没有删除其他文件，用户可以放心实现该功能。

void test\_tool()

{

file\_manager\_tool fmt;

int input;

do

{

menu();

std::string rubbish;

cout << "请输入选项：";

cin >> input;

getline(cin, rubbish);

switch (input)

{

case 1:

fmt.scanner();

break;

case 2:

fmt.show\_copy();

break;

case 3:

fmt.delete\_by\_name();

break;

case 4:

fmt.delete\_by\_md5();

break;

case 5:

fmt.delete\_by\_match();

break;

case 6:

fmt.delete\_all\_copy();

break;

case 0:

break;

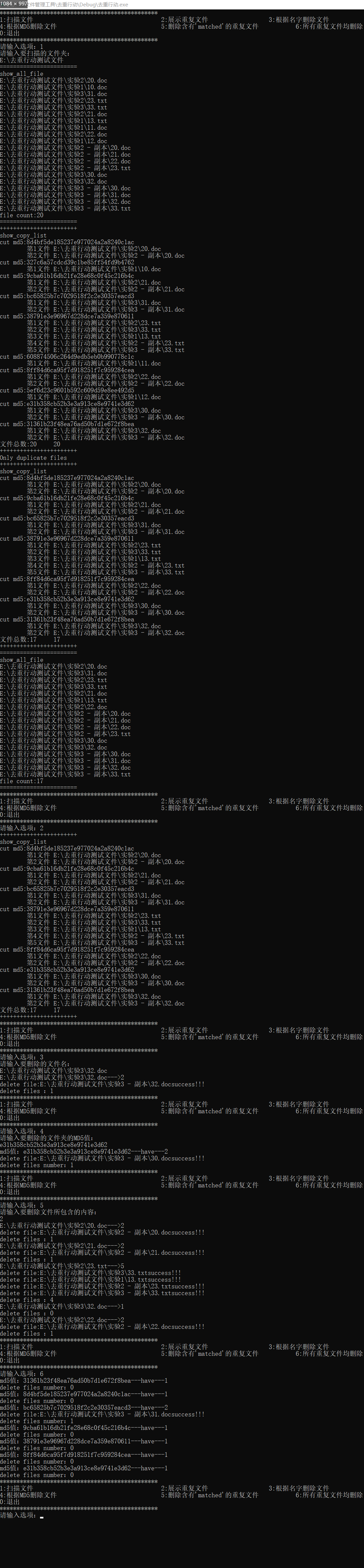
default:

break;

}

} while (input != 0);

}



实现了磁盘管理工具的扫描文件、展示重复文件、根据文件名删除文件、根据MD5值删除文件、删除含有关键字的重复文件、删除所有重复文件这些功能，且经过多次调试，并没有出现错误的，用户可以相应实现

1. **活动主要内容**

根据前期的项目需求，进行编写代码，然后将初步实现的项目发给已经确定的需求人员，让用户进行初步的使用，然后根据用户所反馈的情况进行适当的修改需求，然后修改代码，最后将修改后的项目重新发给已确定需求人，如此反复。最终得到现在实现的项目。

1. **个人完成部分概述**
2. **实现内容**

1）扫描文件

2）文件管理文件

3）测试工具文件

4）测试文件

1. **实现时遇到的问题**

在实现项目时，本着方便用户的想法，当用户点开程序时就默认从C盘开始扫描并进行删除重复文件直至F盘结束，但这个工程量太大，所程序在很长时间内都没有扫描结束，很影响用户体验，所以最后改为由用户指定文件进行扫描，并删除的时候也有用户自己选择删除方式。还有就是在刚开始删除文件时，只显示文件已删除，并没有显示哪些文件删除了，还有哪些文件未删除，最后改为每次删除后都将删除文件名打印一遍。

**八、项目总结**

#### 本次项目设计到此已经顺利结束，通过这次的项目设计,我学到了很多。在以前的三年时间内我学习了很多的课程及编程语言，虽然通过学习对这些课程有所了解和掌握，但是始终处于理论知识的掌握阶段，并没有真正应用于实践设计开发，所以对知识的掌握还不够全面和熟练。而且并不能真正的开发出一个完整的应用于实际的系统。通过这次的项目设计，我和小组同学共同完成了这个项目，对自身的能力也有了很大的提高。

整个项目的过程对我来说是一次能力的真正提高的过程；是一次将理论应用于实践的过程；是将以前所学知识充分利用的过程，是一次真正的实践过程。

总体说来，在这次长达十周的项目设计中，我真正学到了很多有用的东西，是一些在课堂内所学不到的知识，收获很大。但是还是在和小组沟通时有许多不能轻易解决的问题，这就需要我们在之后的学习生活中多去和别人合作来慢慢的熟悉团体合作的过程，以来进一步提高自己的能力。