软件工程

软件测试文档

程序等价确认和判断工具

姓名：王少杰
学号：201220116

2022.12.15

目录

1 静态分析报告 **3**

1.1 静态分析工具的选取及安装 . . . 3

1.2 静态分析工具的使用 . . . 3

1.3 静态分析工具的分析结果 . . . 4

1.4 静态分析工具的修改 . . . 6

2 导引（必读） **7**

2.1 单元测试模块必读 . . . 7

2.2 集成测试模块必读 . . . 7

2.3 代码导引 . . . 7

3 单元测试报告 **8**

3.1 测试说明 . . . 8

3.1.1 测试目的 . . . 8

3.1.2 测试对象 . . . 8

3.1.3 测试环境 . . . 8

3.1.4 测试工具 . . . 8

3.2 单元测试-E-Commend-check-model . . . 9

3.2.1 测试目的 . . . 9

3.2.2 测试用例 . . . 9

3.2.3 预期输出-实际输出 . . . 10

3.2.4 测试覆盖率 . . . 10

3.2.5 结果分析 . . . 11

3.3 单元测试-F-Input-file-model . . . 12

3.3.1 测试目的 . . . 12

3.3.2 测试用例 . . . 12

3.3.3 预期输出-实际输出 . . . 12

3.3.4 测试覆盖率 . . . 13

3.3.5 结果分析 . . . 13

3.4 单元测试-G-Input-data-model . . . 14

3.4.1 测试目的 . . . 14

3.4.2 测试用例 . . . 14

3.4.3 预期输出-实际输出 . . . 14

3.4.4 测试覆盖率 . . . 15

3.4.5 结果分析 . . . 15

3.5 单元测试-K-DSU-model . . . 16

3.5.1 测试目的 . . . 16

3.5.2 测试用例 . . . 16

3.5.3 预期输出-实际输出 . . . 16

3.5.4 测试覆盖率 . . . 17

3.5.5 结果分析 . . . 17

3.6 单元测试-J-Method-model . . . 18

3.6.1 测试目的 . . . 18

3.6.2 测试用例 18
3.6.3 预期输出-实际输出 18
3.6.4 测试覆盖率 19
3.6.5 结果分析 19
3.7 单元测试-L-Output-model 20
3.7.1 测试目的 20
3.7.2 测试用例 20
3.7.3 预期输出-实际输出 20
3.7.4 测试覆盖率 21
3.7.5 结果分析 21

4 集成测试报告 22
4.1 集成测试分析 22
4.2 集成测试一 23
4.2.1 测试目的 23
4.2.2 测试用例 23
4.2.3 预期输出和实际输出 24
4.2.4 结果分析 24
4.3 集成测试二 25
4.3.1 测试目的 25
4.3.2 测试用例 25
4.3.3 预期输出和实际输出 26
4.3.4 结果分析 26
4.4 集成测试三 27
4.4.1 测试目的 27
4.4.2 测试用例 27
4.4.3 预期输出和实际输出 28
4.4.4 结果分析 28
4.5 集成测试四 29
4.5.1 测试目的 29
4.5.2 测试用例 29
4.5.3 预期输出和实际输出 29
4.5.4 结果分析 30

1 静态分析报告

1.1 静态分析工具的选取及安装

我使用的静态分析工具是：Clang Static Analyzer，安装的过程如下：

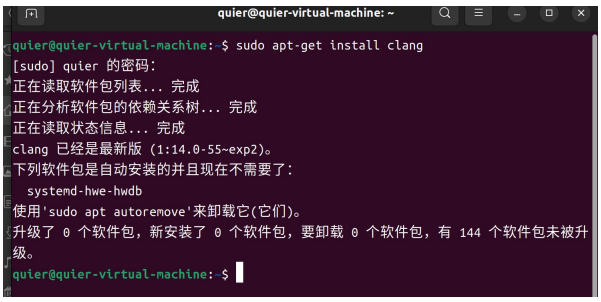


图 1: 安装 clang

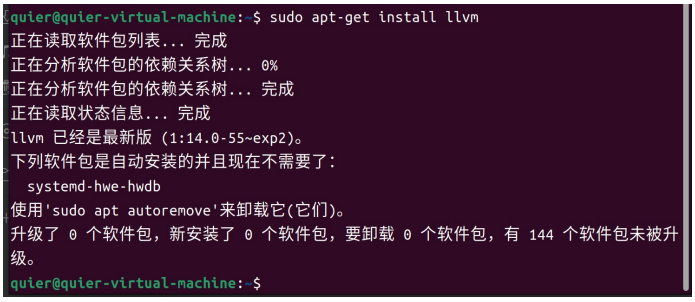
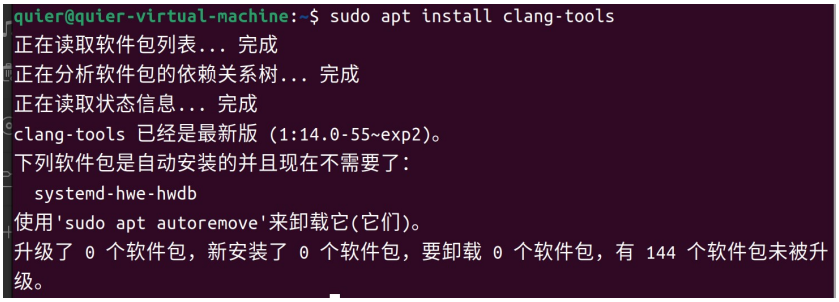


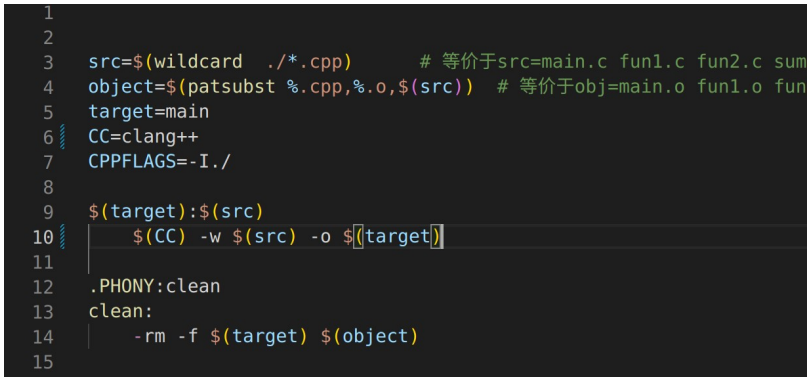
图 2: 安装 LLVM 编译器



安装 scan-build

1.2 静态分析工具的使用

使用 scan-build 首先必须要用 clang 或者 g++ 进行编译，首先使用 make 进行编译。将 makefile 里面的编译器换为 clang++ 编译器。



```makefile


src=$(wildcard  ./*.cpp)      # 等价于src=main.c fun1.c fun2.c sum
object=$(patsubst %.cpp,%.o,$(src))  # 等价于obj=main.o fun1.o fun
target=main
CC=clang++
CPPFLAGS=-I./

$(target):$(src)
    $(CC) -w $(src) -o $(target)

.PHONY:clean
clean:
    -rm -f $(target) $(object)

```

改变编译器

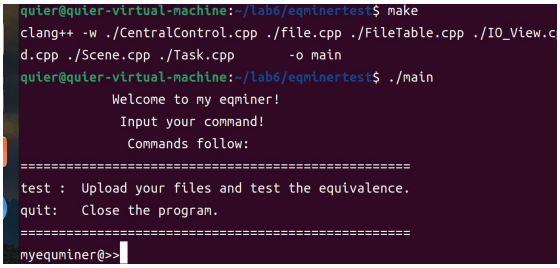


图 3: make 命令，编译程序

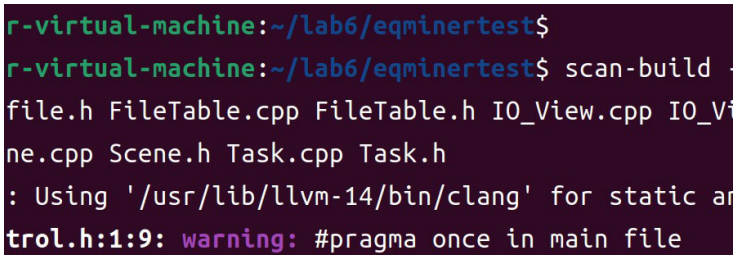
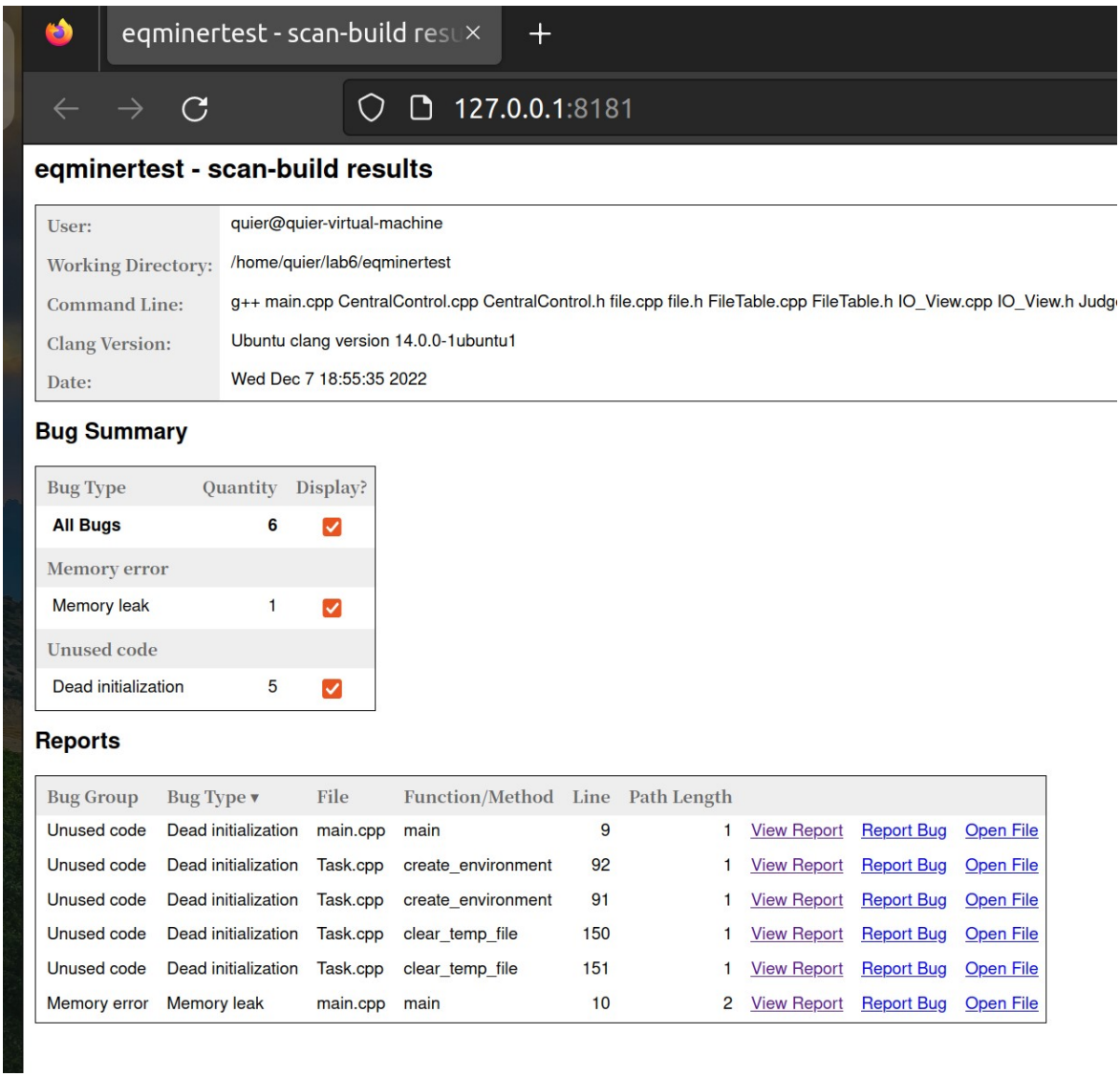


图 4: 输入 scan-build 命令，开始检测

1.3 静态分析工具的分析结果



eqminertest - scan-build results

User:	quier@quier-virtual-machine
Working Directory:	/home/quier/lab6/eqminertest
Command Line:	g++ main.cpp CentralControl.cpp CentralControl.h file.cpp file.h FileTable.cpp FileTable.h IO_View.cpp IO_View.h Judg
Clang Version:	Ubuntu clang version 14.0.0-1ubuntu1
Date:	Wed Dec 7 18:55:35 2022

Bug Summary

Bug Type	Quantity	Display?
All Bugs	6	☑
Memory error		
Memory leak	1	☑
Unused code		
Dead initialization	5	☑

Reports

Bug Group	Bug Type ▾	File	Function/Method	Line	Path Length			
Unused code	Dead initialization	main.cpp	main	9	1	View Report	Report Bug	Open File
Unused code	Dead initialization	Task.cpp	create_environment	92	1	View Report	Report Bug	Open File
Unused code	Dead initialization	Task.cpp	create_environment	91	1	View Report	Report Bug	Open File
Unused code	Dead initialization	Task.cpp	clear_temp_file	150	1	View Report	Report Bug	Open File
Unused code	Dead initialization	Task.cpp	clear_temp_file	151	1	View Report	Report Bug	Open File
Memory error	Memory leak	main.cpp	main	10	2	View Report	Report Bug	Open File

检测结果的网页。有两种错误，一个是内存泄漏，一个是变量未使用。

```
void Judge::create_environment()
{
    //3
    for(int i=0;i<fordernames.size();i++)
    {
        std::string path1=tempprefix+fordernames[i];
        std::string path2=answerfix+fordernames[i];
        int isCreate1 = mkdir(path1.c_str(),S_IRWXU);
        int isCreate2 = mkdir(path2.c_str(),S_IRWXU);
```

Value stored to 'isCreate2' during its initialization is never read

```
        /*
        if( !isCreate1 )
            std::cout<<"create path1: "+path1<<std::enc
```

图 5: 点击 View Report，查看错误的地方。这是 Dead initialization 的错误类型，表示这个变量定义了但是后来并没有访问或者使用。

Show only relevant lines Show control flow arrows

```
1  #include<iostream>
2  #include "CentralControl.h"
3
4
5  int main()
6  {
7          CentralControl my_equminer;
8          my_equminer.start();
9          int *a=new int[9];
```

1 Memory is allocated →

```
10         return 0;
```

2 ← Potential leak of memory pointed to by 'a'

图 6: 查看 main.cpp 里面的内存泄露错误，这个是我自己可以写的，看一下能否检测出来。

Summary > Report 5b2aab

Bug Summary

File: Task.cpp
Warning: line 92, column 13
Value stored to 'isCreate2' during its initialization is never read

Report Bug

Annotated Source Code

Press '?' to see keyboard shortcuts

点击 Report Bug，查看报错。

1.4 静态分析工具的修改

```
void Judge::create_environment()
{
    //3
    for(int i=0;i<fordernames.size();i++)
    {
        std::string path1=tempprefix+fordernames[i];
        std::string path2=answerfix+fordernames[i];
        int isCreate1 = mkdir(path1.c_str(),S_IRWXU);
        int isCreate2 = mkdir(path2.c_str(),S_IRWXU);
        /*
        if( !isCreate1 )
            std::cout<<"create path1: "+path1<<std::en
        else
```

图 7: 修改前：标红圈的两个语句需要被我们注释，因为它们后来没有使用。

```
void Judge::create_environment()
{
    //3
    for(int i=0;i<fordernames.size();i++)
    {
        std::string path1=tempprefix+fordernames[i];
        std::string path2=answerfix+fordernames[i];
        //int isCreate1 = mkdir(path1.c_str(),S_IRWX
        //int isCreate2 = mkdir(path2.c_str(),S_IRWX
        /*
        if( !isCreate1 )
```

图 8: 修改后：标红圈的两个语句被我们注释。

```
quier@quier-virtual-machine:~/lab6/eqminertest$ scan-build -V
 file.cpp file.h FileTable.cpp FileTable.h IO_View.cpp IO_View
thod.h Scene.cpp Scene.h Task.cpp Task.h
scan-build: Using '/usr/lib/llvm-14/bin/clang' for static anal
scan-build: Analysis run complete.
scan-build: Removing directory '/tmp/scan-build-2022-12-07-191
scan-build: No bugs found.
quier@quier-virtual-machine:~/lab6/eqminertest$
```

同时我们又将那个数组（内存泄漏）删除，再一次使用 scan-build 进行检测，没有 bug，所以没有生成报告。

2 导引（必读）

因为本次实验内容繁杂，单元测试模块之间比较雷同，集成测试之间也比较雷同，所以在这一节说明重要的章节，其他可以选择阅览。

2.1 单元测试模块必读

测试说明 说明了单元测试的基本信息。

单元测试-E-Commend-check-model 在覆盖率部分，解释了对于分支覆盖的理解。

2.2 集成测试模块必读

集成测试分析 说明了集成测试的基本信息。

2.3 代码导引

请比对测试树来看，测试树在单元测试报告的测试说明里面。

-input-model 输入模块

—**Command-check-model-test** 命令检测模块

—**file-model-test** 文件对数据结构的检测模块

—**init-data-test** 文件对检测模块

—**input-file-model-test** 子文件输入检测模块

-judge-model 判断模块

—**DSU-model-test** 并查集检测模块

—**method-model-test** 判断等价性检测模块

-output-model 输出模块

—**output-data-test** 文件输出检测模块

-integrationtest 集成测试模块

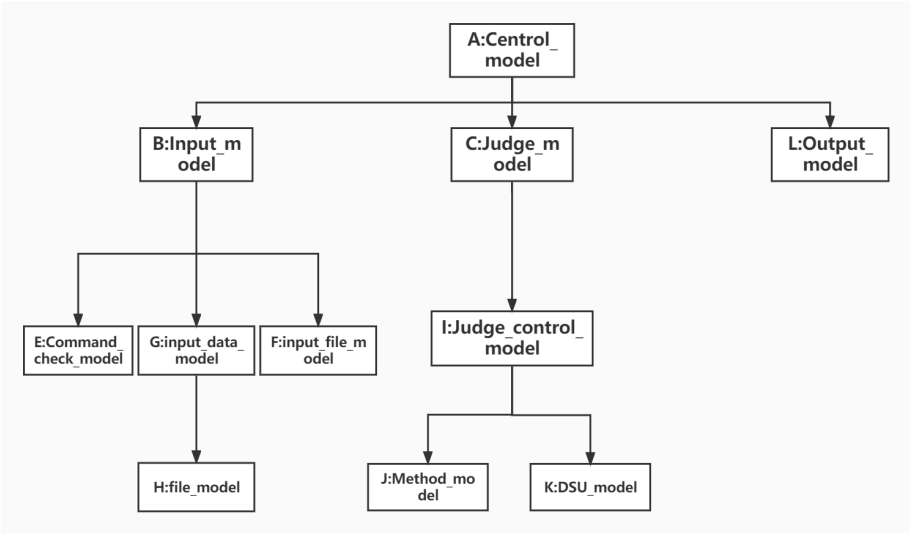
一堆模块的组合，以序号来命名，序号见测试树。

3 单元测试报告

3.1 测试说明

3.1.1 测试目的

本模块主要是测试程序中每一个单元模块是否正确，单元选取的是重要的类或者是函数，分析程序过后，形成了以下的测试图：(下面将以这个图的形式展示单元测试)



3.1.2 测试对象

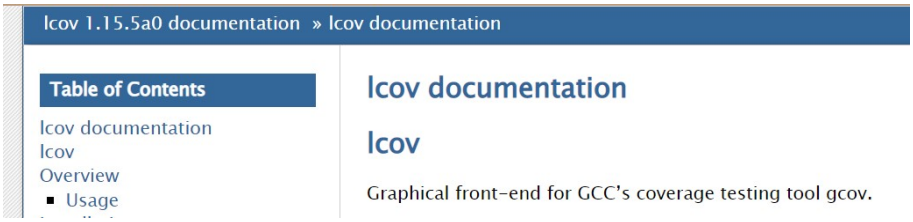
测试程序：实验四，程序等价性判断工具。测试对象：上图中的每一个节点。

3.1.3 测试环境

测试平台：Workstation Beta 虚拟机
操作系统：Ubuntu 64 位

3.1.4 测试工具

测试框架：Gtest 框架
覆盖率分析：lcov 工具包结合 gcov 工具.（都采用语句覆盖和分支覆盖）



3.2 单元测试-E-Commend-check-model

3.2.1 测试目的

本单元主要测试的是用户输入的命令是否的到了正确的解释以及文件输入格式是否正确：

用户命令	程序行为
”test filepath”	判定 test 类型，要进行判定
”quit”	退出程序
回车	打印下一行提示符
其他命令	未知指令

程序运行行为

文件格式	是否合规
若干 cpp 加一个 txt	符合
两个以上 txt	不符合
有其他文件后缀	不符合

文件规范

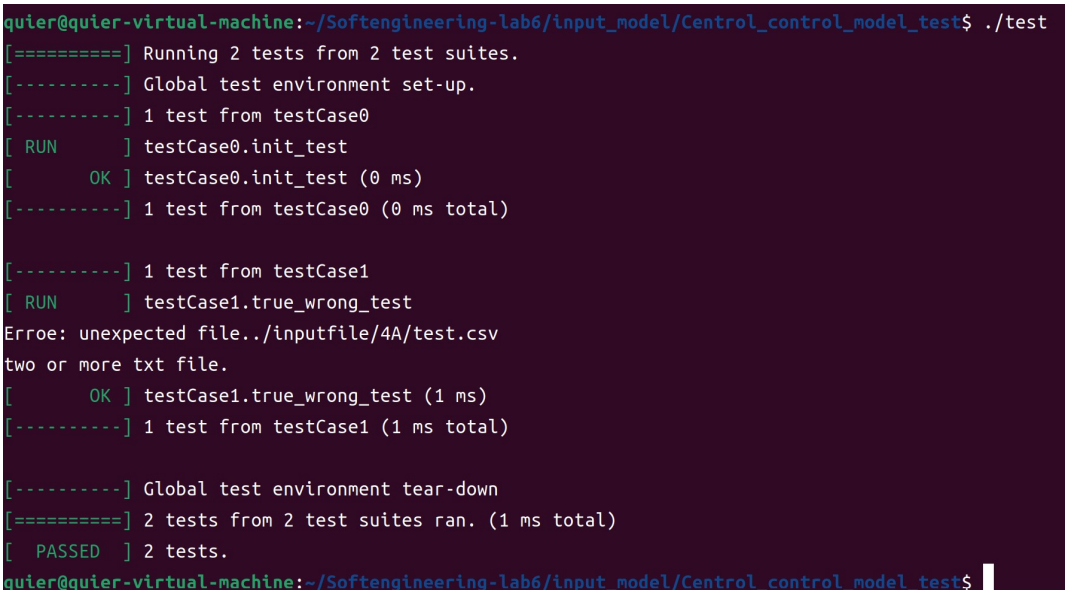
3.2.2 测试用例

```
//测试一
TEST(testCase0,init_test)
{
        string s1="";
        EXPECT_EQ(Command_March(s1),1);//测试输入了回车
        //test command
        string s2="test ./inputfile/50A/123.cpp";
        EXPECT_EQ(Command_March(s2),test);//测试test命令
        string s3="test   ./inputfile/50A/23123.cpp";
        EXPECT_EQ(Command_March(s3),test);//测试test命令
        //quit command
        string s4="quit";
        EXPECT_EQ(Command_March(s4),quit);//测试退出命令
        //unknowncommand
        string s5="assadsadsadsa";
        EXPECT_EQ(Command_March(s5),unknown);//测试未知命令
        string s6=" quit";
        EXPECT_EQ(Command_March(s6),unknown);//测试未知命令
}
//测试二
TEST(testCase1,true_wrong_test)
{
        //right
        //符合文件格式
        EXPECT_EQ(Test_file("./inputfile/50A"),true);
```

```
            //other file
            //拥有其他后缀文件
            EXPECT_EQ(Test_file("./inputfile/4A"),false);
            //double or more txt
            //有两个以上的txt
            EXPECT_EQ(Test_file("./inputfile/51A"),false);
    }
```

3.2.3 预期输出-实际输出

预期是测试程序全部通过，实际输出如下：



```
quier@quier-virtual-machine:~/Softengineering-lab6/input_model/Centrol_control_model_test$ ./test
[==========] Running 2 tests from 2 test suites.
[----------] Global test environment set-up.
[----------] 1 test from testCase0
[ RUN      ] testCase0.init_test
[       OK ] testCase0.init_test (0 ms)
[----------] 1 test from testCase0 (0 ms total)

[----------] 1 test from testCase1
[ RUN      ] testCase1.true_wrong_test
Erroe: unexpected file../inputfile/4A/test.csv
two or more txt file.
[       OK ] testCase1.true_wrong_test (1 ms)
[----------] 1 test from testCase1 (1 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 2 test suites ran. (1 ms total)
[  PASSED  ] 2 tests.
quier@quier-virtual-machine:~/Softengineering-lab6/input_model/Centrol_control_model_test$
```

3.2.4 测试覆盖率

测试一使用分支覆盖，测试二使用语句覆盖/分支覆盖

分支覆盖 分支覆盖就是程序中的每一个分支的真和假都被执行过一次。下面说明测试原理：一部分被测试源代码大致如下：

```

    int Command_March(std::string &command) {
            if (command == "")//分支一
                    ......
            if (test_order == "test ")//分支二
                    ......
            else if (quit_order == "quit")//分支三
                    ......
            else//分支四
                    ......
    }
```

测试命令	分支一	分支二	分支三	分支四
输入回车测试	覆盖 true	未覆盖	未覆盖	未覆盖
test 检测命令	覆盖 false	覆盖 true	未覆盖	未覆盖
quit 命令	覆盖 false	覆盖 false	覆盖 true	未覆盖
其他命令	覆盖 false	覆盖 false	覆盖 false	覆盖了 true

分支覆盖情况

因为最后是 else 语句，没法覆盖 false，覆盖 false 也没意义。

覆盖的报告截图如下：

Filename	Line Coverage			Functions		Branches	
command_test.cpp		100.0 %	22 / 22	100.0 %	9 / 9	50.0 %	9 / 18
test_one.h		100.0 %	12 / 12	100.0 %	1 / 1	100.0 %	8 / 8
test_two.h		88.0 %	22 / 25	100.0 %	1 / 1	89.5 %	17 / 19

覆盖率报告

```
18                :            :         };
19                :            :
20                :          6 : int Command_March(std::string &command)
21                :            : {
22        [ +  + ]:          6 :         if (command == "")
23                :            :         {
24                :          1 :                 return myempty;
25                :            :         }
26        [ +  + ]:          5 :         if (command.substr(0, 5) == "test ")
27                :            :         {
28                :          2 :                 int loc=5;
29        [ +  + ]:          4 :                 while(command[loc]==' ')
30                :          2 :                         loc++;
31                :          2 :                 command = command.substr(loc, command.length() - loc);
32                :          2 :                 return test;
33                :            :         }
34        [ +  + ]:          3 :         else if (command.substr(0, 4) == "quit")
35                :            :         {
36                :          1 :                 return quit;
37                :            :         }
38                :            :         else
39                :          2 :                 return unknown;
40                :            : }
41                :            :
```

色彩化显示

3.2.5 结果分析

通过了本地的测试，并且分支覆盖率达到了 80% 以上，分别为上面的 100% 和 89.5%, 后者没有覆盖为 100%, 经过分析认为是样例没有设计全面，仍有部分分支未被覆盖。

3.3 单元测试-F-Input-file-model

3.3.1 测试目的

因为实验四一个文件夹下面有多个子文件夹，所以本单元目的是检测：文件相关路径是否初始化正确，子文件名是否加载正确。

3.3.2 测试用例

```

        Judge judge("./inputfile");

        Judge judge1("./aaaaaaa");

        //测试一
        TEST(testCase0,filenames_test){
                //50A和4A文件夹名字是否加载正确
                std::vector<std::string> filesname=judge.get_filenames();
                std::vector<std::string> filesnamestandard={"50A","4A"};
                ASSERT_EQ(filesname.size(),filesnamestandard.size());
                for(int i=0;i<filesnamestandard.size();i++)
                        EXPECT_EQ(filesname[i],filesnamestandard[i]);
                std::vector<std::string> filesname1=judge1.get_filenames();
                EXPECT_EQ(filesname1.size(),0);
        }
                //测试二
        TEST(testCase1,status_test){
                //判断完成的状态，此时并未进行判断
                EXPECT_EQ(judge.is_finished(),false);
        }
        //测试三
        TEST(testCase1,filepath_test){
                //查看一些判断需要的中间文件是否生成正确
                EXPECT_EQ(judge.get_sourcefix(),"./inputfile/");
                EXPECT_EQ(judge.get_tempprefix(),"./temp/task/");
                EXPECT_EQ(judge.get_answerfix(),"./answer/task/");
        }
```

3.3.3 预期输出-实际输出

预期是全部通过测试，测试结果如下：

```
[ RUN      ] testCase0.filenames_test
[       OK ] testCase0.filenames_test (0 ms)
[----------] 1 test from testCase0 (0 ms total)

[----------] 2 tests from testCase1
[ RUN      ] testCase1.status_test
[       OK ] testCase1.status_test (0 ms)
[ RUN      ] testCase1.filepath_test
[       OK ] testCase1.filepath_test (0 ms)
[----------] 2 tests from testCase1 (0 ms total)

[----------] Global test environment tear-down
[==========] 3 tests from 2 test suites ran. (0 ms total)
[  PASSED  ] 3 tests.
quier@quier-virtual-machine:~/Softengineering-lab6/input_model/inp
```

3.3.4 测试覆盖率

Filename	Line Coverage			Functions		Branches	
input_file_test.cpp		100.0 %	20 / 20	100.0 %	13 / 13	54.5 %	12 / 22
test.h		100.0 %	20 / 20	100.0 %	6 / 6	100.0 %	10 / 10

可以看到语句覆盖是 100%, 分支覆盖是 100%.

```
40                :            :     //1
41                :          2 :         std::string task = "task/";
42                :          2 :         tempprefix = "./temp/" + task;
43                :          2 :         answerfix = "./answer/" + task;
44                :          2 :         finished = false;
45                :            :
46                :            :     //2
47                :            :         DIR* pDir;   //  是头文件<dirent.h>的类型
48                :            :         struct dirent* ptr;  // opendir、readdir这些都是头文件di
49        [ +  + ]:          2 :         if (!(pDir = opendir(sourcepath.c_str()))) return;
50                :            :
51        [ +  + ]:          5 :         while ((ptr = readdir(pDir)) != 0)
52                :            :         {
53  [ +  +  +  + ]:          4 :             if (strcmp(ptr->d_name, ".") != 0 && strcmp(pt
54                :            :             {
55                :          2 :                 std::string filename = ptr->d_name;
56                :          2 :             fordernames.push_back(filename);
57                :          2 :             }
58                :            :         }
59                :          1 :         closedir(pDir);
60                :            :
61        [ +  + ]:          2 : }
```

色彩化显示

3.3.5 结果分析

两种覆盖率都是 100%，并且测试通过。说明这个模块能够正确的将子文件夹的名字读入，创建每一个子文件夹的路径，为后面的等价判断模块做出准备。

3.4 单元测试-G-Input-data-model

3.4.1 测试目的

测试文件对是否被真正加载进内存当中，设计的策略是等程序运行完了以后，再用驱动程序读出来并与已初始化好的文件对进行对比。(因为 file-model 十分简单, 所以融合了，这本属于集成测试)

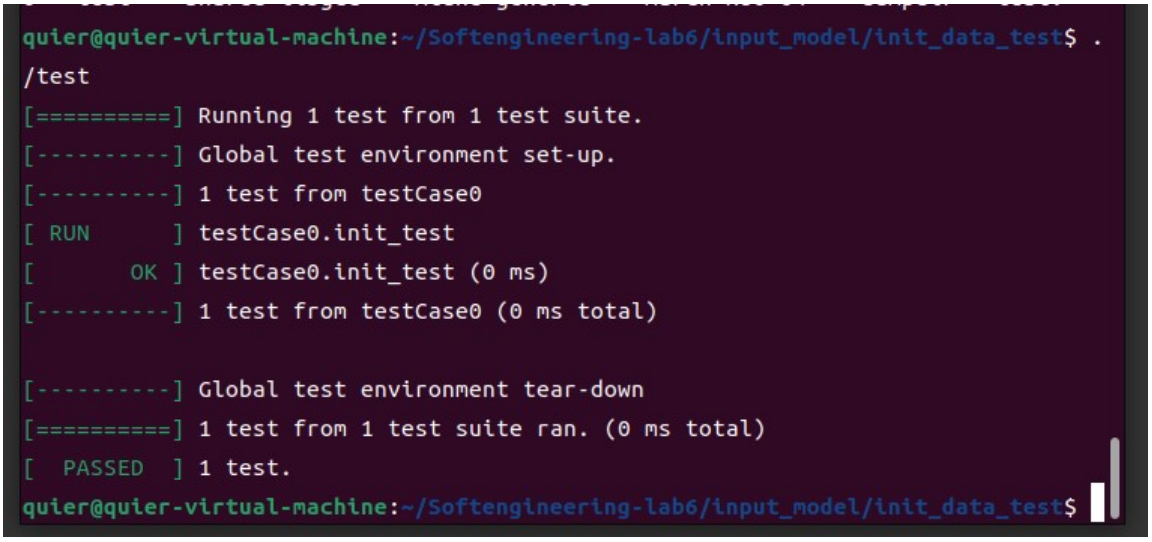
3.4.2 测试用例

因为样例繁多，所以展示关键点。

```

        //初始化要读入的文件，以后用于比对。
            std::vector<string> files={
                "inputfile/50A/138805414.cpp",
                "inputfile/50A/142890373.cpp",
                "inputfile/50A/164831265.cpp",
            };
                //读入文件对
                string answerpath="./answer/task/50A";
                StackFileTable pt;
                pt.init_data("inputfile/50A",answerpath);

                //读取文件对，进行比较url1，url2就是files数组的元素。
                pt.Get_file_pairs(s1,s2);
                EXPECT_EQ(s1,url1);
                EXPECT_EQ(s2,url2);
```

3.4.3 预期输出-实际输出

预期是全部通过测试，测试结果如下：



测试结果

3.4.4 测试覆盖率

Filename	Line Coverage			Functions		Branches	
file.cpp		40.0 %	2 / 5	50.0 %	1 / 2	100.0 %	2 / 2
file.h		100.0 %	5 / 5	100.0 %	1 / 1	-	0 / 0
init_data_test.cpp		100.0 %	26 / 26	100.0 %	5 / 5	50.0 %	11 / 22
test.cpp		100.0 %	25 / 25	100.0 %	3 / 3	83.3 %	15 / 18

Generated by: LCOV version 1.15

主要测试的文件是 test.cpp, 可以看到 test.cpp 文件的语句覆盖率达到了 100%，分支覆盖率达到了 83.3%。

```
20                  :           : }
21                  :         1 : void StackFileTable::init_data(std::string s,std::string answer)
22                  :           : {
23                  :           :
24                  :           :         //1
25                  :         1 :         answerfile=answer;
26                  :           :         //2
27                  :         1 :         std::string path = s;
28                  :           :         DIR *pDir;   //  是头文件<dirent.h>的类型
29                  :           :     struct dirent *ptr;  // opendir、readdir这些都是头文件dirent.h
30                  :         1 :         std::vector<std::string>filesvec;
31          [ - + ]:         1 :     if (!(pDir = opendir(path.c_str()))) return;
32          [ + + ]:         7 :     while ((ptr = readdir(pDir)) != 0) {
33                  :           :         // strcmp是C语言里的，只导入string,然后std::strcmp都是没有的
34  [ + + + + ]:         6 :         if (strcmp(ptr->d_name, ".") != 0 && strcmp(ptr->d_name, ".
35                  :         4 :                         std::string filename=ptr->d_name;
36          [ + + ]:         4 :                         if(filename.substr(filename.length()-4,4)!=
37                  :         3 :                     filesvec.push_back(path + "/" + ptr->d_name);  // 可
38                  :         4 :         }
39                  :           :     }
40                  :         1 :     closedir(pDir);
41          [ + + ]:         4 :         for (int i = 0; i < filesvec.size(); i++)
42                  :           :         {
43          [ + + ]:         6 :                 for (int j = i + 1; j < filesvec.size(); j++)
44                  :           :                 {
45                  :         3 :                         Add_file_pairs(filesvec[i], filesvec[j]);
46                  :           :                 }
47                  :           :         }
48  [ + - + - ]:         1 : }
```

lcov 报告色彩化显示 test.cpp 主要部分

3.4.5 结果分析

测试顺利通过，但是分支覆盖率没有 100%, 主要原因是因为上面图片的第 31 行是判断文件 50A 和 4A 是否是正确的文件路径设置的，但其实在上一个单元测试里面就已经测试过这里了，这里文件路径肯定是正确的。最后一个 48 行应该是 lcov 自己的检测问题。

3.5 单元测试-K-DSU-model

3.5.1 测试目的

主要检查并查集的对错,union 函数和 find 函数是否正确。

3.5.2 测试用例

因为样例繁多，所以展示主要测试方法。

```

        //首先初始化并查集，构造函数输入是一个string的vector
        files.push_back("inputfile/50A/30534178.cpp");//three
        files.push_back("inputfile/50A/1111111111.cpp");//four
        files.push_back("inputfile/50A/2222222222.cpp");//five
        DSU dsu3(files);

        // zero connect one
        // cover else { // rank[pRoot] == rank[qRoot]
        //执行union操作
        dsu3.unionElements("inputfile/50A/142890373.cpp",
        "inputfile/50A/45851050.cpp");

        //最后通过find操作来判断对错
    EXPECT_EQ(dsu3.isConnected("inputfile/50A/45851050.cpp",
    "inputfile/50A/30534178.cpp"),false);
```

3.5.3 预期输出-实际输出

预期是全部通过测试，测试结果如下：

```
quier@quier-virtual-machine:~/Softengineering-lab6/judge_model/DSU_model_test$ ./test
[==========] Running 3 tests from 3 test suites.
[----------] Global test environment set-up.
[----------] 1 test from testCase0
[ RUN      ] testCase0.init_test
[       OK ] testCase0.init_test (0 ms)
[----------] 1 test from testCase0 (0 ms total)

[----------] 1 test from testCase1
[ RUN      ] testCase1.connected_test
[       OK ] testCase1.connected_test (0 ms)
[----------] 1 test from testCase1 (0 ms total)

[----------] 1 test from testCase2
[ RUN      ] testCase2.rank_test_one
[       OK ] testCase2.rank_test_one (0 ms)
[----------] 1 test from testCase2 (0 ms total)
```

测试结果

3.5.4 测试覆盖率

Filename	Line Coverage			Functions		Branches	
dsu_test.cpp		100.0 %	43 / 43	100.0 %	13 / 13	50.0 %	9 / 18
test.h		96.9 %	31 / 32	100.0 %	4 / 4	81.2 %	13 / 16

Generated by: LCOV version 1.15

主要测试的文件是 test.cpp, 可以看到 test.cpp 文件的语句覆盖率达到了 96.9%，分支覆盖率达到了 81.2%。

```
:          :    }
:        5 :     void unionElements(std::string s1, std::string s2)
:          :    {
:          :
:        5 :        int p = path_to_index[s1];
:        5 :        int q = path_to_index[s2];
:        5 :        int pRoot = find(p);
:        5 :        int qRoot = find(q);
:          :
:        5 :        if (pRoot == qRoot)
:        0 :            return;
:          :
:        5 :        if (rank[pRoot] < rank[qRoot]) {
:        1 :            parent[pRoot] = qRoot;
:          :        }
:        4 :        else if (rank[qRoot] < rank[pRoot]) {
:        1 :            parent[qRoot] = pRoot;
:          :        }
:          :        else { // rank[pRoot] == rank[qRoot]
:        3 :            parent[pRoot] = qRoot;
:        3 :            rank[qRoot] += 1;
:          :        }
:          :    }
:          : };
```

lcov 报告色彩化显示 test.cpp 主要部分（union 函数）

3.5.5 结果分析

分支覆盖没有全面是因为上面那个图中的 return 没有覆盖，这个只有在 union(A,A) 这种错误情况下才会产生，但是前面的 init-data-model 模块已经排除了这个情况，所以无需担心。本单元总体测试过关。

3.6 单元测试-J-Method-model

3.6.1 测试目的

本模块是直接判断程序是否等价的模块。本模块测试了许多小的模块，包括：输入格式生成模块、随机数生成模块、判断内容是否相等模块、综合判断模块。我们要测试是否能生成合规的数据并准确判断等价性。

3.6.2 测试用例

因为样例繁多，所以这里展示判断内容是否相等模块样例

```

        //将要判断的文件都初始化好
        std::string  s4 ="./inputfile/50A/stdin_format.txt";
        vector<std::string> files4{
                "./inputfile/50A/138805414.cpp",
                "./inputfile/50A/142890373.cpp",
                "./inputfile/50A/164831265.cpp",
        };
        //创建判断类
        Sample_Test * pt4=new Sample_Test();
        //初始化
        pt4->init(s4,files4,"./mytemp/50A");
        pt4->init_exe_file(files4);

        //使用judge方法判断
         bool test1=pt4->judge("./inputfile/50A/138805414.cpp"
         ,"./inputfile/50A/142890373.cpp");
         //测试正确性
        EXPECT_EQ(test1,false);
```

3.6.3 预期输出-实际输出

预期是全部通过测试，测试结果如下：

```
[----------] 1 test from testCase2
[ RUN      ] testCase2.Compare_result_test
[       OK ] testCase2.Compare_result_test (3 ms)
[----------] 1 test from testCase2 (3 ms total)

[----------] 1 test from testCase3
[ RUN      ] testCase3.load_Executablefile_test
[       OK ] testCase3.load_Executablefile_test (5453 ms)
[----------] 1 test from testCase3 (5453 ms total)

[----------] Global test environment tear-down
[==========] 4 tests from 4 test suites ran. (5459 ms total)
[  PASSED  ] 4 tests.
```

测试结果

3.6.4 测试覆盖率

Filename	Line Coverage			Functions		Branches	
method_test.cpp		99.1 %	112 / 113	100.0 %	20 / 20	58.3 %	49 / 84
test.cpp		94.6 %	140 / 148	100.0 %	9 / 9	83.3 %	40 / 48
test.h		81.2 %	13 / 16	57.1 %	4 / 7	100.0 %	6 / 6

Generated by: LCOV version 1.15

主要测试的文件是 test.cpp, 可以看到 test.cpp 文件的语句覆盖率达到了 94.6%，分支覆盖率达到了 83.3%。

```
 96                :          3 : bool Sample_Test::judge(std::string s1, std::string
 97                :            : {
 98                :          3 :         int label1=filename_to_index[s1];
 99                :          3 :         int label2=filename_to_index[s2];
100          [ - + ]:          3 :         if(compile_error[label1]|compile_error[lab
101                :          0 :                 return false;
102                :            :
103                :          6 :         std::string redirect1 = tempfiles[label1] +
104                :          6 :         std::string redirect2 = tempfiles[label2] +
105                :            :
106                :            :
107                :            :
108          [ + + ]:         28 :         for (int i = 0; i < judge_epoch; i++)
109                :            :         {
110                :         27 :                 create_test_data();
111                :         27 :                 int code_one_error2=system(redirec
112                :         27 :                 int code_two_error2=system(redirec
113                :         27 :                 bool is_equal=compare_test_result(
114          [ + + ]:         27 :                 if (!is_equal)
115                :          2 :                         return false;
116                :            :         }
117                :            :
118                :            :
119                :          1 :         return true;
120                :          3 : }
```

lcov 报告色彩化显示 test.cpp 主要部分（等价判断函数）

3.6.5 结果分析

测试成功，并且覆盖率达到了要求。经过分析，上面的图中有一个分支没有覆盖到是因为我设计的样例没有涉及到程序编译出问题的情况，所以后续检测中可以加入一些编译有误程序来覆盖这一个分支。

3.7 单元测试-L-Output-model

3.7.1 测试目的

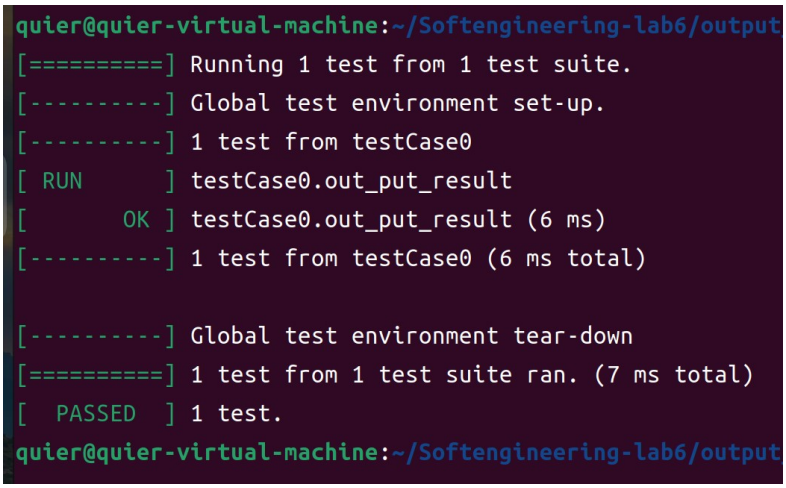
本模块测试已经判断完的文件对是否真正的被输出到 csv 文件当中。

3.7.2 测试用例

```

//首先写驱动模块，初始化文件对
StackFileTable table;
std::string url1="./inputfile/50A/11111111.cpp";
std::string url2="./inputfile/50A/22222222.cpp";
table.Set_answer(url1,url2,true);

//执行要被判断的模块
table.save_file_result();

//读取csv文件，判断是否输出正确
std::string test_url1="inputfile/50A/11111111.cpp";
std::string test_url2="inputfile/50A/22222222.cpp";

fstream file;//读取csv
file.open("./answer/50A/equal.csv");
file>>test_str;
EXPECT_EQ(test_str,test_url1+","+test_url2);//判断函数
file.close();
```

3.7.3 预期输出-实际输出

预期是全部通过测试，测试结果如下：



测试结果

3.7.4 测试覆盖率

Filename	Line Coverage			Functions		Branches	
FileTable.cpp		96.7 %	29 / 30	100.0 %	3 / 3	92.9 %	13 / 14
FileTable.h		14.3 %	1 / 7	14.3 %	1 / 7	-	0 / 0
file.cpp		100.0 %	5 / 5	100.0 %	2 / 2	100.0 %	2 / 2
file.h		100.0 %	6 / 6	100.0 %	2 / 2	-	0 / 0
out_put_test.cpp		100.0 %	40 / 40	100.0 %	5 / 5	50.0 %	6 / 12

Generated by: LCOV version 1.15

主要测试的文件是 FileTable.cpp, 可以看到 FileTable.cpp 文件的语句覆盖率达到了 96.7%，分支覆盖率达到了 92.9%。

```
75                :           :        // 向csv文档中写入数据
76                :         1 :    std::ofstream dataFile_eq;
77                :         1 :        dataFile_eq.open(answerfile+"/equal.csv"
78                :           :
79                :         1 :        std::ofstream dataFile_uneq;
80                :         1 :        dataFile_uneq.open(answerfile+"/inequal.
81                :           :
82        [ +  + ]:         7 :        while (!outputfile.empty())
83                :           :        {
84                :        12 :                std::string url1=turn_to_erlativ
85                :        12 :                std::string url2=turn_to_erlativ
86                :         6 :                bool state=outputfile.top().Get_
87        [ +  + ]:         6 :                if(state)
88                :         1 :                        dataFile_eq <<url1<<","<
89                :           :                else
90                :         5 :                        dataFile_uneq <<url1<<",
91                :         6 :                outputfile.pop();
92                :         6 :        }
93                :         1 :        dataFile_eq.close();
94                :         1 :        dataFile_uneq.close();
95                :         1 : }
```

lcov 报告色彩化显示 FileTable.cpp 主要部分（保存成 csv 格式函数）

3.7.5 结果分析

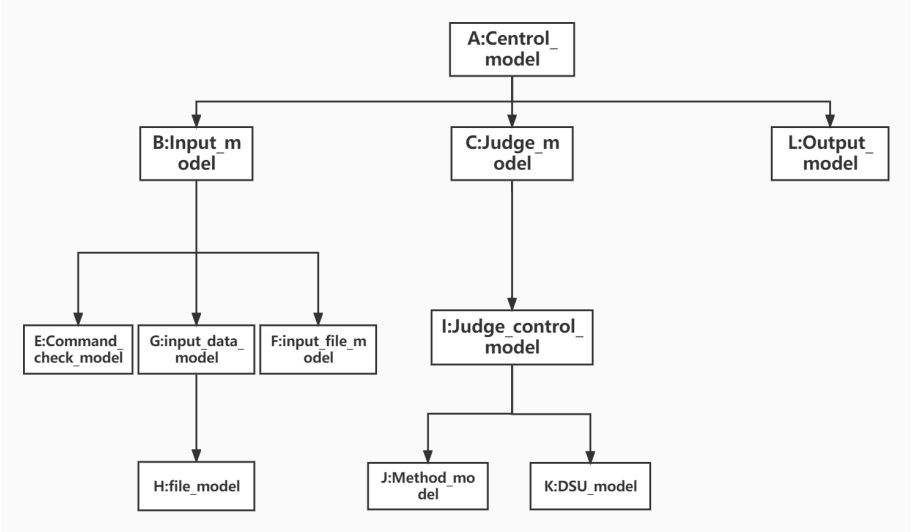
测试成功，并且覆盖率达到了要求。

```
                :           : std::string turn_to_erlativepath(std::string s)
                :           : {
                :        12 :        int times=3;
        [ +  - ]:       324 :        for(int i=s.size()-1;i>=0;i--)
                :           :        {
+  +  +  +      :       324 :                if(s[i]=='/'&& times==1)
      +  + ]
                :           :                {
                :        12 :                        return s.substr(i+1,s.size()-i);
                :           :                }
        [ +  + ]:       312 :                else if(s[i]=='/')
                :        24 :                  times--;
                :           :        }
                :         0 :        return "";
```

这里没有覆盖是因为不可能执行到 return，这个函数的作用是将路径转化为相对路径，因为我们的文件肯定是有相对路径的（前面模块保证了路径正确性）。这里只是为了通过编译，所以返回了一个空串。

4 集成测试报告

4.1 集成测试分析



项目测试树

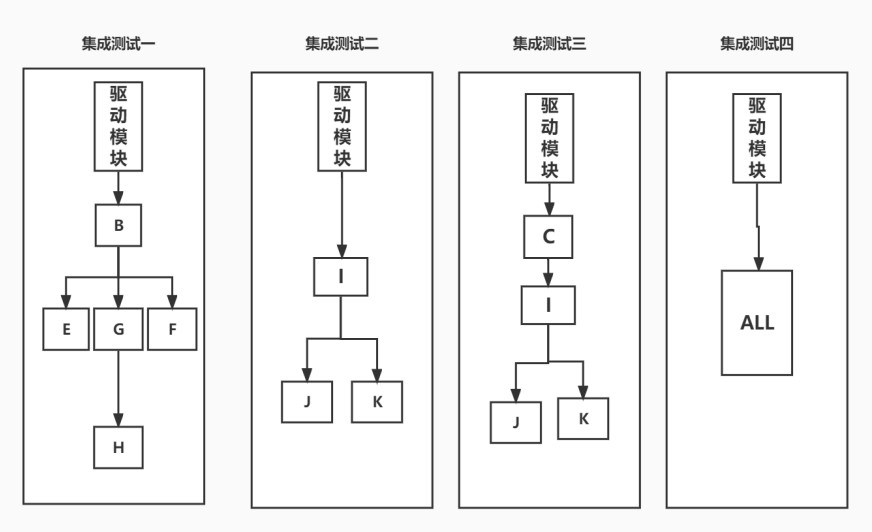
测试目的 因为测试树的每一个节点都在单元测试部分完成了，所以现在我们把它拼起来，检查它们之间的交互有没有什么问题。

测试工具 gtest 框架 +lcov 工具

测试对象 单元测试中单元的组合。

测试环境 测试平台：Workstation Beta 虚拟机
操作系统：Ubuntu 64 位

测试方法 自底而上的测试方法。所以我结合测试树画了下面的测试步骤。



集成测试步骤图（结合测试树）

4.2 集成测试一

4.2.1 测试目的

这个模块主要是测试输入模块，具体说就是: 用户输入测试命令，程序内部保存好程序对，以供后面的判断模块使用。

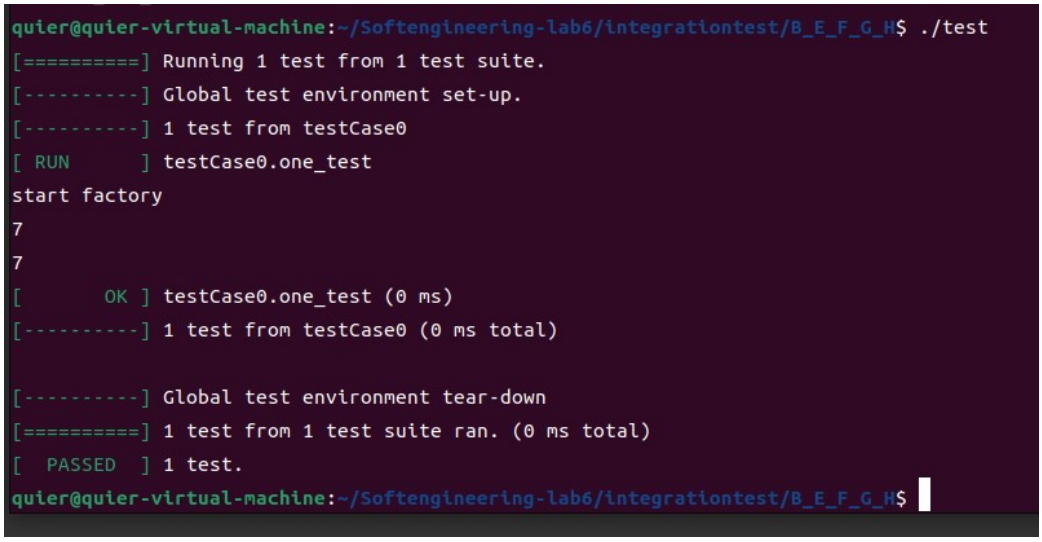
4.2.2 测试用例

```
        //一共进行两组测试
         test_for_50A(my_equminer);
         test_for_4A(my_equminer);
      //其中一组测试
      void test_for_50A(my_equminer){
      //第一步: 执行被测试的模块
      //test_start函数模拟了用户输入, 相当于用户输入了“test ./inputfile”
      CentralControl * my_equminer=new CentralControl();
      my_equminer->test_start("test ./inputfile");

      //第二步: 创建测试的样例(存在set中), 也就是匹配的文件对。
      test_file_group.insert({"./inputfile/4A/84822638.cpp",
              "./inputfile/4A/48762087.cpp"});
      test_file_group.insert({"./inputfile/4A/84822639.cpp",
              "./inputfile/4A/48762087.cpp"});
      test_file_group.insert({"./inputfile/4A/101036360.cpp",
              "./inputfile/4A/48762087.cpp"});

      //第三步: 进行测试
   while (!pt1->is_empty()){

      pt1->Get_file_pairs(url1,url2);//获取文件对

      //判断是否在set中
      bool flag=(test_file_group.find({url1,url2})!=test_file_group.end())||
      (test_file_group.find({url2,url1})!=test_file_group.end());

      //不在说明测试失败
      EXPECT_EQ(flag,true);
      //在了就将这个文件对删去
      test_file_group.erase({url1,url2});
      test_file_group.erase({url2,url1});
   }
    //最后如果程序生成的等价对正确, 那么test_file_group应该全部删完了。
    //此时set大小应该为0, 再检查一下。
    EXPECT_EQ(test_file_group.size(),0);
```

```
    }
```

4.2.3 预期输出和实际输出

预期输出就是测试全部通过，显示绿色，实际输出如下图：



测试结果

4.2.4 结果分析

从结果分析，测试用例通过了说明用户输入 test 命令之后，内部数据结构已经成功存储了文件对，说明输入过程已经完成。

4.3 集成测试二

4.3.1 测试目的

这个模块主要是测试每一个子文件夹的判断是否正确，具体测试的是：程序能否正确判断一个子文件夹内的等价性关系。

4.3.2 测试用例

```
        //一共进行两组测试
    test_for_one();
    test_for_two();
    //其中一组测试
    test_for_two(){
        //第一步：数据准备阶段。准备等价的程序对
        std::set<std::string> inequal_test={
                "inputfile/4A/84822639.cpp,inputfile/4A/48762087.cpp",
                "inputfile/4A/84822639.cpp,inputfile/4A/101036360.cpp",
                "inputfile/4A/84822639.cpp,inputfile/4A/117364748.cpp",
                "inputfile/4A/84822638.cpp,inputfile/4A/48762087.cpp",
        }
        //第二步：跑等价判断程序
        JudgeControl * c=new JudgeControl(sourcefile1,answerfile1,tempfile1);
        c->start_judge();
        //第三步：读取inequal.csv文件，与inequal_test进行对比。
        //读取文件
        fstream file;
        file.open("./answer/Task/4A/inequal.csv");
        string answer_str;
        for(int i=0;i<9;i++){
                file>>answer_str;
                //进行对比
                bool flag=inequal_test.find(answer_str)!=inequal_test.end();
                //进行判断
                EXPECT_EQ(flag,true);
                //从set中删除文件对
                inequal_test.erase(answer_str);
        }
        //最后如果全部匹配正确，那么set大小一定为0
        EXPECT_EQ(inequal_test.size(),0);
        file.close();

        //equal.csv也是一样的。
        ......
    }
```

4.3.3 预期输出和实际输出

预期输出就是测试全部通过，显示绿色，实际输出如下图：

```
./inputfile/50A/21508887.cpp ./inputfile/50A/21508898.cpp
./inputfile/50A/21508887.cpp ./inputfile/50A/29019948.cpp
ss./inputfile/50A/21508887.cpp ./inputfile/50A/29019948.cpp
./inputfile/50A/21508887.cpp ./inputfile/50A/21715601.cpp
ss./inputfile/50A/21508887.cpp ./inputfile/50A/21715601.cpp
[       OK ] testCase0.one_test (3385 ms)
[----------] 1 test from testCase0 (3385 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (3385 ms total)
[  PASSED  ] 1 test.
quier@quier-virtual-machine:~/Softengineering-lab6/integrationtest/I_J_K$
```

测试结果

4.3.4 结果分析

测试用例全部通过，说明程序正确判断了程序的等价性，判断过程得到了检验。

4.4 集成测试三

4.4.1 测试目的

这个模块主要是测试整体判断模块，具体说就是: 检查所有的子文件夹是否都判断正确。

4.4.2 测试用例

```

        //第一步：跑Judge模块，会直接判断多个子文件夹。
        Judge judge("./inputfile");
        judge.start_factories();
        judge.save_results();

        //会进行两组测试。
        //test one
        test_for_4A();
        //test for two
        test_for_50A();

        //第二步：数据准备。
        //储存所有的等价程序对
        std::set<std::string> equal_test={
                "inputfile/50A/21508887.cpp,inputfile/50A/21715601.cpp",
                "inputfile/50A/21508887.cpp,inputfile/50A/29019948.cpp",
                "inputfile/50A/21508887.cpp,inputfile/50A/21508898.cpp",
                "inputfile/50A/21715601.cpp,inputfile/50A/29019948.cpp",
                ......
                }
        //读取等价csv进行判断，测试逻辑同集成测试二
        fstream file;
        file.open("./answer/task/50A/equal.csv");
        string answer_str;
        for(int i=0;i<6;i++)
        {
                file>>answer_str;
                bool flag=equal_test.find(answer_str)!=equal_test.end();
                EXPECT_EQ(flag,true);
                equal_test.erase(answer_str);
        }
        EXPECT_EQ(equal_test.size(),0);
        file.close();
    }
```

4.4.3 预期输出和实际输出

预期输出就是测试全部通过，显示绿色，实际输出如下图：

```
quier@quier-virtual-machine:~/Softengineering-lab6/integrationtest/C_I_J_K$ ./test
[==========] Running 1 test from 1 test suite.
[----------] Global test environment set-up.
[----------] 1 test from testCase0
[ RUN      ] testCase0.one_test
int(1,16)
int(1,16)
int(1,100)
[       OK ] testCase0.one_test (2714 ms)
[----------] 1 test from testCase0 (2714 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (2714 ms total)
[  PASSED  ] 1 test.
quier@quier-virtual-machine:~/Softengineering-lab6/integrationtest/C_I_J_K$
```

测试结果

4.4.4 结果分析

测试通过，说明程序可以同时处理多个子文件夹，多个子文件夹都可以判断正确。

4.5 集成测试四

4.5.1 测试目的

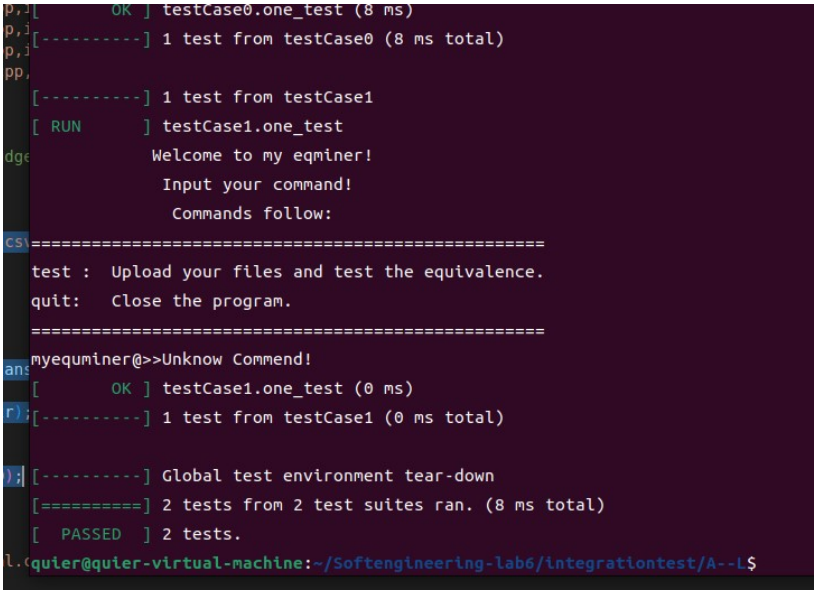
测试整个软件，检查用户输入检测命令后，程序能否输出正确的两个 csv 文件。

4.5.2 测试用例

```

        //第一步，模拟用户输入
        CentralControl my_equminer;
        my_equminer.DeviceDriver_Test("test ./inputfile");

        //第二步，准备数据（下面以equal为例），存储在set中
        std::set<std::string> equal_test={
                "inputfile/50A/21508887.cpp,inputfile/50A/21715601.cpp",
                "inputfile/50A/21508887.cpp,inputfile/50A/29019948.cpp",
                "inputfile/50A/21508887.cpp,inputfile/50A/21508898.cpp",
                ......
                };
        //第三步，读取equal.csv。进行检测
        fstream file;
        file.open("./inputfile/equal.csv");
        string answer_str;
        file>>answer_str;
        for(int i=0;i<7;i++)
        {
                //读取文件对
                file>>answer_str;
                //判断在不在set中
                bool flag=equal_test.find(answer_str)!=equal_test.end();
                //assert判断
                ASSERT_EQ(flag,true);
                //删除匹配到的文件对
                equal_test.erase(answer_str);
        }
        file.close();
        //如果全部匹配正确，那么最后equal_test大小就会变为0。
        //再检测一下是否等于0
        ASSERT_EQ(equal_test.size(),0);
```

4.5.3 预期输出和实际输出

预期输出就是测试全部通过，显示绿色，实际输出如下图：



测试结果

4.5.4 结果分析

测试通过，软件整体流程没有问题。整个流程是指：输入测试指令，程序解析指令，程序存储文件对，程序判断文件对，以及最后的程序输出 csv 文件。