lab6 实验报告

一、静态分析

由于使用C++进行编程，选择CPPCheck进行静态分析.

安装

```
          VirtualBox:~$ sudo apt install cppcheck
[sudo] ~    的密码:
正在读取软件包列表... 完成
正在分析软件包的依赖关系树
正在读取状态信息... 完成
下列软件包是自动安装的并且现在不需要了:
  linux-hwe-5.4-headers-5.4.0-100 linux-hwe-5.4-headers-5.4.0-126
  linux-hwe-5.4-headers-5.4.0-128 linux-hwe-5.4-headers-5.4.0-131
  linux-hwe-5.4-headers-5.4.0-84
使用'sudo apt autoremove'来卸载它(它们)。
建议安装:
  cppcheck-gui
下列【新】软件包将被安装:
  cppcheck
升级了 0 个软件包，新安装了 1 个软件包，要卸载 0 个软件包，有 30 个软件包未被升级。
有 1 个软件包没有被完全安装或卸载。
需要下载 0 B/921 kB 的归档。
解压缩后会消耗 3,660 kB 的额外空间。
正在选中未选择的软件包 cppcheck。
(正在读取数据库 ... 系统当前共安装有 270523 个文件和目录。)
正准备解包 .../cppcheck_1.82-1_amd64.deb  ...
正在解包 cppcheck (1.82-1) ...
正在设置 cppcheck (1.82-1) ...
```

1. 分析等价判断工具

使用CPPCheck对第四次实验完成的等价判断工具进行分析，结果如下图

```
          /irtualBox:~/Software Engineering/lab4$ cppcheck --enable=all lab4-code/ > result.txt
[lab4-code/data_structure.cpp:192]: (style) The scope of the variable 'ran' can be reduced.
[lab4-code/data_structure.cpp:31]: (warning) Member variable 'folder::_file_num' is not initialized in the constructor.
[lab4-code/data_structure.cpp:8]: (performance) Function parameter 'path' should be passed by reference.
[lab4-code/data_structure.cpp:8]: (performance) Function parameter 'name' should be passed by reference.
[lab4-code/data_structure.cpp:31]: (performance) Function parameter 'path' should be passed by reference.
[lab4-code/data_structure.cpp:31]: (performance) Function parameter 'name' should be passed by reference.
[lab4-code/data_structure.cpp:114]: (performance) Function parameter 'path' should be passed by reference.
[lab4-code/data_structure.cpp:114]: (performance) Function parameter 'name' should be passed by reference.
[lab4-code/judge.cpp:33] -> [lab4-code/judge.cpp:42]: (style) Variable 'b2' is reassigned a value before the old one has been used.
[lab4-code/judge.cpp:31]: (style) The scope of the variable 'b2' can be reduced.
[lab4-code/main.cpp:31]: (error) Memory leak: fd
[lab4-code/preprocess.cpp:5]: (performance) Function parameter 'path' should be passed by reference.
[lab4-code/preprocess.cpp:5]: (performance) Function parameter 'name' should be passed by reference.
```

第一处`data_structure.cpp`中`ran`变量的作用域可以缩小，修改后如下

```
//int ran = 0;
char ch;
while (!success) {
    int ran = 0;
    ran = distribute(generator);
    if (ran < 91 || ran > 96) {
        ch = ran;
        success = 1;
    }
}
```

第二处`data_structure.cpp`中`folder`类构造函数未初始化成员`file_num`，修改后如下

```
folder::folder(string path, string name)
    : _path(path)
    , _name(name)
    , _file_num(0) {}
```

第三处将一些构造函数中`string`值传递改为引用传递

```
file(string &path, string &name);
```

```
testcase(string &path, string &name);
```

第四处将`judge.cpp`中`b2`变量放入循环中声明，修改后如下

```
//bool b2 = 0;
for(int i = 0;i < num;i++){
    //b2 = 1;
    string cmd = command;       // cmd =
    cmd += testpath;            // cmd =
    cmd += to_string(i);        // cmd =
    cmd += ".txt >";            // cmd =
    cmd += dirname;             // cmd =
    cmd += "/";                 // cmd =
    cmd += to_string(i);        // cmd =
    cmd += ".txt";              // cmd =
    bool b2 = system(cmd.c_str());
```

第五处将`main`函数中存在空指针，有内存泄漏问题，进行修改后如下

```
//folder* fd =  new folder{};
folder* fd = pre->init_folder();
```

修改完成后再次检查，结果如下

```
           VirtualBox:~/Software Engineering/lab4$ cppcheck --enable=style lab4-code/
Checking lab4-code/data_structure.cpp ...
1/5 files checked 20% done
Checking lab4-code/judge.cpp ...
2/5 files checked 40% done
Checking lab4-code/main.cpp ...
3/5 files checked 60% done
Checking lab4-code/output.cpp ...
4/5 files checked 80% done
Checking lab4-code/preprocess.cpp ...
5/5 files checked 100% done
```

2. 分析等价确认工具

使用CPPCheck对第五次实验完成的等价确认工具进行分析，结果如下图

```
Software Engineering/lab5$ cppcheck --enable=style confirmingqt/
Checking confirmingqt/file.cpp ...
[confirmingqt/file.h:19]: (style) Class 'file' has a constructor with 1 argument that is not explicit.
1/9 files checked 11% done
Checking confirmingqt/initial.cpp ...
[confirmingqt/initial.cpp:40]: (style) The scope of the variable 'f1' can be reduced.
[confirmingqt/initial.cpp:40]: (style) The scope of the variable 'f2' can be reduced.
2/9 files checked 22% done
Checking confirmingqt/main.cpp ...
3/9 files checked 33% done
Checking confirmingqt/menu.cpp ...
4/9 files checked 44% done
Checking confirmingqt/moc_file.cpp ...
Checking confirmingqt/moc_file.cpp: Q_MOC_OUTPUT_REVISION...
5/9 files checked 55% done
Checking confirmingqt/moc_initial.cpp ...
Checking confirmingqt/moc_initial.cpp: Q_MOC_OUTPUT_REVISION...
6/9 files checked 66% done
Checking confirmingqt/moc_menu.cpp ...
Checking confirmingqt/moc_menu.cpp: Q_MOC_OUTPUT_REVISION...
7/9 files checked 77% done
Checking confirmingqt/moc_result.cpp ...
Checking confirmingqt/moc_result.cpp: Q_MOC_OUTPUT_REVISION...
8/9 files checked 88% done
Checking confirmingqt/result.cpp ...
9/9 files checked 100% done
```

第一处`file`类中构造函数仅含一个参数，改为显式声明

```
public:
    explicit file(QObject *parent = nullptr);
    file();
    explicit file(string &path);
```

第二处将`initial.cpp`中变量`f1,f2`作用域减小

```
    //bool f1, f2;

    while(!equ.eof()){
        equ >> str;
        bool f1 = regex_match(str, m1, r);
        if(f1){
            ve.push_back(make_pair(m1.str(1), m1.str(2)));
        }
        else cout << "Not match." << endl;
    }
    while(!inequ.eof()){
        inequ >> str;
        bool f2 = regex_match(str, m2, r);
        if(f2){
            vine.push_back(make_pair(m2.str(1), m2.str(2)));
        }
        else cout << "Not match." << endl;
    }
}
```

修改完成后再次检查，结果如下

```
           VirtualBox:~/Software Engineering/lab5$ cppcheck --enable=style confirmingqt/
Checking confirmingqt/file.cpp ...
1/9 files checked 11% done
Checking confirmingqt/initial.cpp ...
2/9 files checked 22% done
Checking confirmingqt/main.cpp ...
3/9 files checked 33% done
Checking confirmingqt/menu.cpp ...
4/9 files checked 44% done
Checking confirmingqt/moc_file.cpp ...
Checking confirmingqt/moc_file.cpp: Q_MOC_OUTPUT_REVISION...
5/9 files checked 55% done
Checking confirmingqt/moc_initial.cpp ...
Checking confirmingqt/moc_initial.cpp: Q_MOC_OUTPUT_REVISION...
6/9 files checked 66% done
Checking confirmingqt/moc_menu.cpp ...
Checking confirmingqt/moc_menu.cpp: Q_MOC_OUTPUT_REVISION...
7/9 files checked 77% done
Checking confirmingqt/moc_result.cpp ...
Checking confirmingqt/moc_result.cpp: Q_MOC_OUTPUT_REVISION...
8/9 files checked 88% done
Checking confirmingqt/result.cpp ...
9/9 files checked 100% done
```

完成了静态分析.

二、单元测试

1. 测试目的

对等价判断工具中的类、对象、方法等进行检查和验证，以保证正确性。

2. 测试对象

对工具的底层数据结构进行测试，集中于`data_structure.cpp`中，对`file`类、`folder`类、`sets`类和`testcase`类的接口与方法进行测试。

3. 测试环境

使用`Ubuntu 18.04.6 LTS`操作系统，gcc version 11.1.0

4. 测试工具

使用GoogleTest工具进行测试，用gcov查看覆盖率。

5. 测试方法

在`UnitTest.cpp`中有5个固件类，均继承于::testing::Test类：`InputFolderTest`、`FolderTest`、`FileTest`、`setsTest`、`testcaseTest`，其中`InputFolderTest`用于测试能否读取Input文件夹中每个子文件夹名，其余4个类用于测试同名类。继承于::testing::Test类中的SetUp方法中，可以实现我们需要构造的数据；继承于::testing::Test类中的TearDown方法中，可以实现一些资源的释放。

输入Input文件夹中有5个子文件夹：`Fd1`、`Fd2`、`Fd3`、`Fd4`、`Fd5`。

首先进行`InputFolderTest`，

```
class InputFolderTest :public::testing::Test
{
protected:
  void SetUp() override {
    fd->read_file_name();
  }
  void TearDown() override {
    delete fd;
  }
  string fd_path = "./";
  string fd_name = "input";
  folder * fd = new folder(fd_path, fd_name);
};
```

初始化folder类即为Input文件夹，调用`read_file_name()`读取文件夹中子文件夹名。

分别测试获取文件夹路径、文件夹名、文件数的接口，期望`get_path()`得到`./`，`get_name()`得到`input`，`get_num()`得到5.

```
TEST_F(InputFolderTest, testGetPath){
    EXPECT_STREQ(fd_path.c_str(), fd->get_path().c_str());
}

TEST_F(InputFolderTest, testGetName){
    EXPECT_STREQ(fd_name.c_str(), fd->get_name().c_str());
}

TEST_F(InputFolderTest, testReadFileNameNum){
    EXPECT_EQ(5, fd->get_num());
}

TEST_F(InputFolderTest, testReadFileName){
    set<string> folder_names;
    for(auto it = 0;it < fd->get_num();it++){
        folder_names.insert(fd->get_file_name(it));
    }
    EXPECT_EQ(1, (int)folder_names.count("Fd1"));
    EXPECT_EQ(1, (int)folder_names.count("Fd2"));
    EXPECT_EQ(1, (int)folder_names.count("Fd3"));
    EXPECT_EQ(1, (int)folder_names.count("Fd4"));
    EXPECT_EQ(1, (int)folder_names.count("Fd5"));
}
```

之后进行`FolderTest`，初始化`Fd1`文件夹，调用`read_file()`方法，读取子文件夹`Fd1`中的所有文件。

```
class FolderTest: public::testing::Test{
protected:
    void SetUp() override {
        fd->read_file();
    }
    void TearDown() override {
        delete fd;
    }

    string fd_path = "./input";
    string fd_name = "Fd1";
    folder * fd = new folder(fd_path, fd_name);
};
```

`Fd1`文件夹下有6个`.cpp`文件，还要测试了是否正确读取文件，测试了类中存放文件名的向量以及类中存放文件的向量中某文件路径。

```
TEST_F(FolderTest, testReadFileNum){
    EXPECT_EQ(6, fd->get_num());
}

TEST_F(FolderTest, testReadFile) {
    set<string> file_names;
    for(auto it = 0;it < fd->get_num();it++){
        file_names.insert(fd->_files[it].get_name());
    }
    string file_path = "./input/Fd1";
    EXPECT_STREQ(file_path.c_str(), fd->_files[0].get_path().c_str());
    EXPECT_EQ(1, (int)file_names.count("108793.cpp"));
    EXPECT_EQ(1, (int)file_names.count("130206.cpp"));
    EXPECT_EQ(1, (int)file_names.count("220805.cpp"));
    EXPECT_EQ(1, (int)file_names.count("281291.cpp"));
    EXPECT_EQ(1, (int)file_names.count("680588.cpp"));
    EXPECT_EQ(1, (int)file_names.count("800017.cpp"));
}
```

其次进行`FileTest`，针对`input/Fd1/108793.cpp`文件进行测试，初始化该文件，读取文件数据。

```
class FileTest: public::testing::Test {
protected:
  void SetUp() override {
    f->read_data();
  }
  void TearDown() override {
    delete f;
  }

  string f_path = "./input/Fd1";
  string f_name = "108793.cpp";
  file* f = new file(f_path, f_name);
};
```

分别测试获取文件路径、文件名、文件类`issame`成员及其相关方法、获取文件数据的接口，期望`get_path()`得到`./input`；`get_name()`得到`108793.cpp`；`get_same()`先为false，调用`set_same()`后变为true；`get_data()`得到该文件的内容。

```
TEST_F(FileTest, testGetPath){
    EXPECT_STREQ(f_path.c_str(), f->get_path().c_str());
}

TEST_F(FileTest, testGetName){
    EXPECT_STREQ(f_name.c_str(), f->get_name().c_str());
}

TEST_F(FileTest, testMemberSame){
    EXPECT_FALSE(f->get_same());
    f->set_same();
    EXPECT_TRUE(f->get_same());
}

TEST_F(FileTest, testGetData){
    ifstream ifs("./input/Fd1/108793.cpp");
    string data((std::istreambuf_iterator<char>(ifs) ), (std::istreambuf_iterator<char>()));
    ifs.close();
    EXPECT_STREQ(data.c_str(), f->get_data().c_str());
}
```

然后进行`setsTest`，初始化一个`sets`类s。

```
class setsTest: public::testing::Test {
protected:
    void SetUp() override {
    }
    void TearDown() override {
        delete s;
    }

    sets* s = new sets();
};
```

测试`resize()`方法与`add_to()`、`find()`方法。初始时s.size()期望值为0，调用`s.resize(3)`后期望值为3。初始时s中0号、1号、2号set全为空调用`find()`方法期望得到false；然后调用`s.add_to(0,1)`、`s.add_to(1,2)`，向0号集合中加入1，向1号集合中加入2，则按照实现逻辑，0号集合中有1、2，1号集合中有0、2，2号集合中有0、1，期望调用`s.find()`，返回值均

为true。

```
TEST_F(setsTest, testResize){
    EXPECT_EQ(0, (int)s->_Sets.size());
    s->_Sets.resize(3);
    EXPECT_EQ(3, (int)s->_Sets.size());
}

TEST_F(setsTest, testInsert){
    s->_Sets.resize(3);
    EXPECT_FALSE(s->find(0, 1));
    EXPECT_FALSE(s->find(2, 0));
    s->add_to(0, 1);
    EXPECT_TRUE(s->find(1, 0));
    s->add_to(1, 2);
    EXPECT_TRUE(s->find(0, 1));
    EXPECT_TRUE(s->find(0, 2));
    EXPECT_TRUE(s->find(1, 0));
    EXPECT_TRUE(s->find(1, 2));
    EXPECT_TRUE(s->find(2, 0));
    EXPECT_TRUE(s->find(2, 1));
}
```

最后进行`testcase`，初始化一个testcase类tc，调用`read_data()`读取stdin_format.txt文件中的格式，调用`read_format()`对读取的内容进行解析。

```
class testcaseTest: public::testing::Test {
protected:
    void SetUp() override {
        tc->read_data();
        tc->read_format();
    }
    void TearDown() override {
        tc->delcases();
        delete tc;
    }
    string testcase_path = "./input/Fd1";
    string testcase_name = "stdin_format.txt";
    testcase* tc = new testcase(testcase_path, testcase_name);
};
```

测试时调用`tc.create(10)`生成十个测试用例，然后读取测试用例检查是否符合

stdin_format.txt文件中的要求，本例中即1-1000间整数。

```
TEST_F(testcaseTest, testCreate){
    tc->create(10);
    for(int i = 0;i < 10;i++){
        string p = "./input/Fd1/testcases/";
        p += to_string(i);
        p += ".txt";
        ifstream ifs(p);
        string data((std::istreambuf_iterator<char>(ifs) ), (std::istreambuf_iterator<char>()));
        ifs.close();
        int number = atoi(data.c_str());
        EXPECT_TRUE(number >= 1);
        EXPECT_TRUE(number <= 1000);
    }
}
```

6. 测试结果分析

在命令行中执行`g++ --coverage -c data_structure.cpp UnitTest.cpp -lgtest -lpthread`

`g++ --coverage data_structure.o UnitTest.o -o UnitTestTest.exe -lgtest -lpthread`
对单元测试代码进行编译，

`./UnitTestTest.exe`运行UnitTest得到结果

```
[==========] Running 13 tests from 5 test suites.
[----------] Global test environment set-up.
[----------] 4 tests from InputFolderTest
[ RUN      ] InputFolderTest.testGetPath
[       OK ] InputFolderTest.testGetPath (0 ms)
[ RUN      ] InputFolderTest.testGetName
[       OK ] InputFolderTest.testGetName (0 ms)
[ RUN      ] InputFolderTest.testReadFileNameNum
[       OK ] InputFolderTest.testReadFileNameNum (0 ms)
[ RUN      ] InputFolderTest.testReadFileName
[       OK ] InputFolderTest.testReadFileName (0 ms)
[----------] 4 tests from InputFolderTest (0 ms total)

[----------] 2 tests from FolderTest
[ RUN      ] FolderTest.testReadFileNum
[       OK ] FolderTest.testReadFileNum (0 ms)
[ RUN      ] FolderTest.testReadFile
[       OK ] FolderTest.testReadFile (0 ms)
[----------] 2 tests from FolderTest (0 ms total)

[----------] 4 tests from FileTest
[ RUN      ] FileTest.testGetPath
[       OK ] FileTest.testGetPath (0 ms)
[ RUN      ] FileTest.testGetName
[       OK ] FileTest.testGetName (0 ms)
[ RUN      ] FileTest.testMemberSame
[       OK ] FileTest.testMemberSame (0 ms)
[ RUN      ] FileTest.testGetData
[       OK ] FileTest.testGetData (0 ms)
[----------] 4 tests from FileTest (0 ms total)

[----------] 2 tests from setsTest
[ RUN      ] setsTest.testResize
[       OK ] setsTest.testResize (0 ms)
[ RUN      ] setsTest.testInsert
[       OK ] setsTest.testInsert (0 ms)
[----------] 2 tests from setsTest (0 ms total)

[----------] 1 test from testcaseTest
[ RUN      ] testcaseTest.testCreate
[       OK ] testcaseTest.testCreate (4 ms)
[----------] 1 test from testcaseTest (4 ms total)

[----------] Global test environment tear-down
```

```
[==========] 13 tests from 5 test suites ran. (6 ms total)
[  PASSED  ] 13 tests.
```

共13条测试用例均通过测试。

之后命令行输入`gcov UnitTest`得到测试覆盖率，

```
File 'UnitTest.cpp'
Lines executed:100.00% of 119
Creating 'UnitTest.cpp.gcov'

File '/usr/local/include/gtest/internal/gtest-internal.h'
Lines executed:90.48% of 21
Creating 'gtest-internal.h.gcov'

File '/usr/include/c++/11/iostream'
No executable lines
Removing 'iostream.gcov'

File '/usr/local/include/gtest/gtest-printers.h'
Lines executed:0.00% of 27
Creating 'gtest-printers.h.gcov'

File '/usr/include/c++/11/ext/new_allocator.h'
Lines executed:68.42% of 19
Creating 'new_allocator.h.gcov'

File '/usr/include/c++/11/bits/stl_tree.h'
Lines executed:78.98% of 157
Creating 'stl_tree.h.gcov'

File '/usr/include/c++/11/bits/allocator.h'
Lines executed:50.00% of 4
Creating 'allocator.h.gcov'
```

```
File '/usr/local/include/gtest/gtest-assertion-result.h'
Lines executed:50.00% of 6
Creating 'gtest-assertion-result.h.gcov'

File '/usr/include/c++/11/bits/charconv.h'
Lines executed:0.00% of 25
Creating 'charconv.h.gcov'

File '/usr/include/c++/11/bits/char_traits.h'
Lines executed:0.00% of 22
Creating 'char_traits.h.gcov'

File '/usr/local/include/gtest/gtest-message.h'
No executable lines
Removing 'gtest-message.h.gcov'

File '/usr/local/include/gtest/internal/gtest-port.h'
Lines executed:0.00% of 1
Creating 'gtest-port.h.gcov'

File '/usr/include/c++/11/new'
Lines executed:0.00% of 2
Creating 'new.gcov'

Lines executed:52.03% of 738
```

统计时有一些问题，统计了较多头文件库函数等内容，截取一部分图片，最终结果为52.03%，应该低于实际测试覆盖率。

三、集成测试

1. 测试目的

集成测试，是在单元测试的基础上，对各单元进行组装，把分离的单元组装为完整的可执行的软件。目的是检查单元部件是否能够集成为一个整体，完成一定的功能，并找出单元测试中没有发现的错误，包括数据定义有没有重合与冲突，接口会不会产生错误，组合以后的模块功能会不会互相影响，组合的系统是不是达到预期的效果等。

2. 测试对象

单元测试已保证了底层数据结构的正确实现，本次将在软件整个运行流程中进行测试。将测试`preprocess.cpp`、`judge.cpp`、`output.cpp`中实现的方法。

3. 测试环境

同单元测试

4. 测试工具

同单元测试

5. 测试方法

在`IntegrationTest.cpp`中存在有1个继承于::testing::Test类的固件类，IntegrationTest。类中将初始化output文件夹和输出文件，为后续测试做准备。

```
class IntegrationTest: public::testing::Test
{
protected:
  void SetUp() override {
    string outpath = "./output";
    if(!filesystem::exists(outpath)) filesystem::create_directory(outpath);
    ofstream equ(outpath + "/equal.csv");
    ofstream inequ(outpath + "/inequal.csv");
    equ << "file1" << "," << "file2" << endl;
    inequ << "file1" << "," << "file2" << endl;
    equ.close();
    inequ.close();
  }
  void TearDown() override {
    delete root;
  }
```

准备了5个输入子文件夹，进行5组集成测试。

`Fd1`文件夹中有6个文件，在`input_Fd1.md`中描述了代码期望解决的问题，仅有`800017.cpp`和`680588.cpp`d等价。

`Fd2`文件夹中有4个文件，输入一个int 范围1-2 和一个 string 长度2-3，若int为1输出Hello。所有文件均完全相同，测试预处理`find_same()`方法。

`Fd3`文件夹中有4个文件，输入一个大写或小写字符char，期望将大小写进行转换。其中`569301.cpp,534432.cpp,744250.cpp`互相等价，`979597.cpp`与他们不等价。

`Fd4`文件夹中是实验4提供的测试用例中的两个不等价文件，检查是否正确处理两个文件的情况。

`Fd5`文件夹中有3个文件，在`input_Fd5.md`中描述了代码期望解决的问题，`680518.cpp,276460.cpp`等价。

在testAll测试中，测试输入子文件夹数量，预期为5。5次循环，每次使用preprocess类`init_folder()`方法初始化文件夹，测试文件夹路径、名、文件内容判断是否成功初始化。使用preprocess类`find_same()`方法找出完全相同的文件，针对`Fd2`进行测试。之后使用testcase类生成测试用例，用judge类进行OJ程序代码编译与运行，得到输出结果，进行比较判断等价。

使用output类将结果写入equal.csv和inequal.csv文件中。读取这两个文件对结果进行测试。且最终总共应有31条等价与不等价结果记录。部分测试用例如下：

```
TEST_F(IntegrationTest, testAll){
    EXPECT_EQ(0, root->get_num());
    root->read_file_name();
    EXPECT_EQ(5, root->get_num());
```

```
if(fdn_name.compare("Fd2") == 0){
    EXPECT_TRUE(sts->find(0, 1));
    EXPECT_TRUE(sts->find(0, 2));
    EXPECT_TRUE(sts->find(0, 3));
    EXPECT_TRUE(sts->find(1, 0));
    EXPECT_TRUE(sts->find(1, 2));
    EXPECT_TRUE(sts->find(1, 3));
    EXPECT_TRUE(sts->find(2, 0));
    EXPECT_TRUE(sts->find(2, 1));
    EXPECT_TRUE(sts->find(2, 3));
    EXPECT_TRUE(sts->find(3, 0));
    EXPECT_TRUE(sts->find(3, 1));
    EXPECT_TRUE(sts->find(3, 2));
}
```

```
judge* judgement = new judge{};
bool exe = judgement->execute(*fd, 10);
EXPECT_TRUE(exe);
```

```
output* result = new output{};
bool success = result->save(*fd, *sts);
EXPECT_TRUE(success);
```

```
EXPECT_EQ(31, equals+inequals);
```

6. 测试结果分析

在命令行中执行`g++ --coverage -c data_structure.cpp preprocess.cpp judge.cpp output.cpp IntegrationTest.cpp -lgtest -lpthread`

```
g++ --coverage data_structure.o preprocess.o judge.o output.o IntegrationTest.o -o IntegrationTest.exe -lgtest -lpthread
```

```
./IntegrationTest.exe
```

成功通过测试

```
[==========] Running 1 test from 1 test suite.
[----------] Global test environment set-up.
[----------] 1 test from IntegrationTest
[ RUN      ] IntegrationTest.testAll
No same files in folder Fd1
compare 800017.cpp & 108793.cpp
compare 800017.cpp & 220805.cpp
compare 800017.cpp & 281291.cpp
compare 800017.cpp & 130206.cpp
compare 800017.cpp & 680588.cpp
compare 108793.cpp & 220805.cpp
compare 108793.cpp & 281291.cpp
compare 108793.cpp & 130206.cpp
compare 108793.cpp & 680588.cpp
compare 220805.cpp & 281291.cpp
compare 220805.cpp & 130206.cpp
compare 220805.cpp & 680588.cpp
compare 281291.cpp & 130206.cpp
compare 281291.cpp & 680588.cpp
compare 130206.cpp & 680588.cpp
No same files in folder Fd3
compare 979597.cpp & 569301.cpp
compare 979597.cpp & 534432.cpp
compare 979597.cpp & 744250.cpp
compare 569301.cpp & 534432.cpp
compare 569301.cpp & 744250.cpp
file 999120.cpp and file 79608.cpp are the same.
file 999120.cpp and file 327182.cpp are the same.
file 999120.cpp and file 129222.cpp are the same.
No same files in folder Fd4
compare 134841308.cpp & 48762087.cpp
No same files in folder Fd5
compare 680518.cpp & 862538.cpp
compare 680518.cpp & 276460.cpp
compare 862538.cpp & 276460.cpp
[       OK ] IntegrationTest.testAll (6602 ms)
[----------] 1 test from IntegrationTest (6602 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (6602 ms total)
[  PASSED  ] 1 test.
```