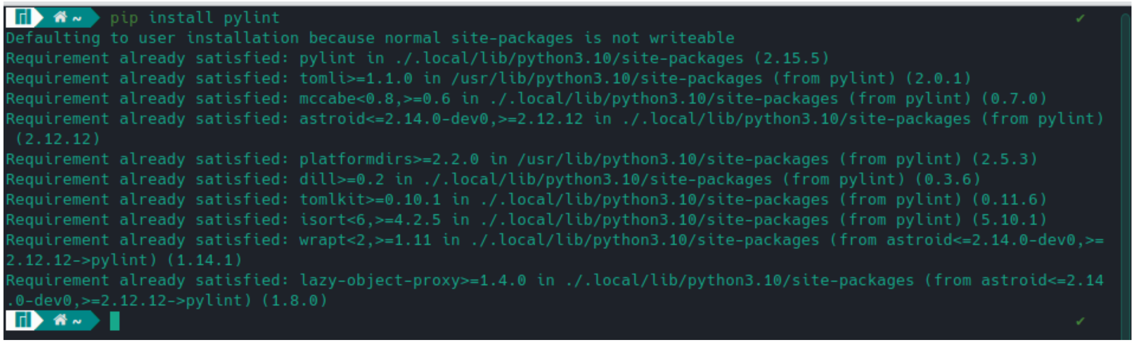
软件工程lab6实验报告

201840058 蒋潇鹏

一、静态分析

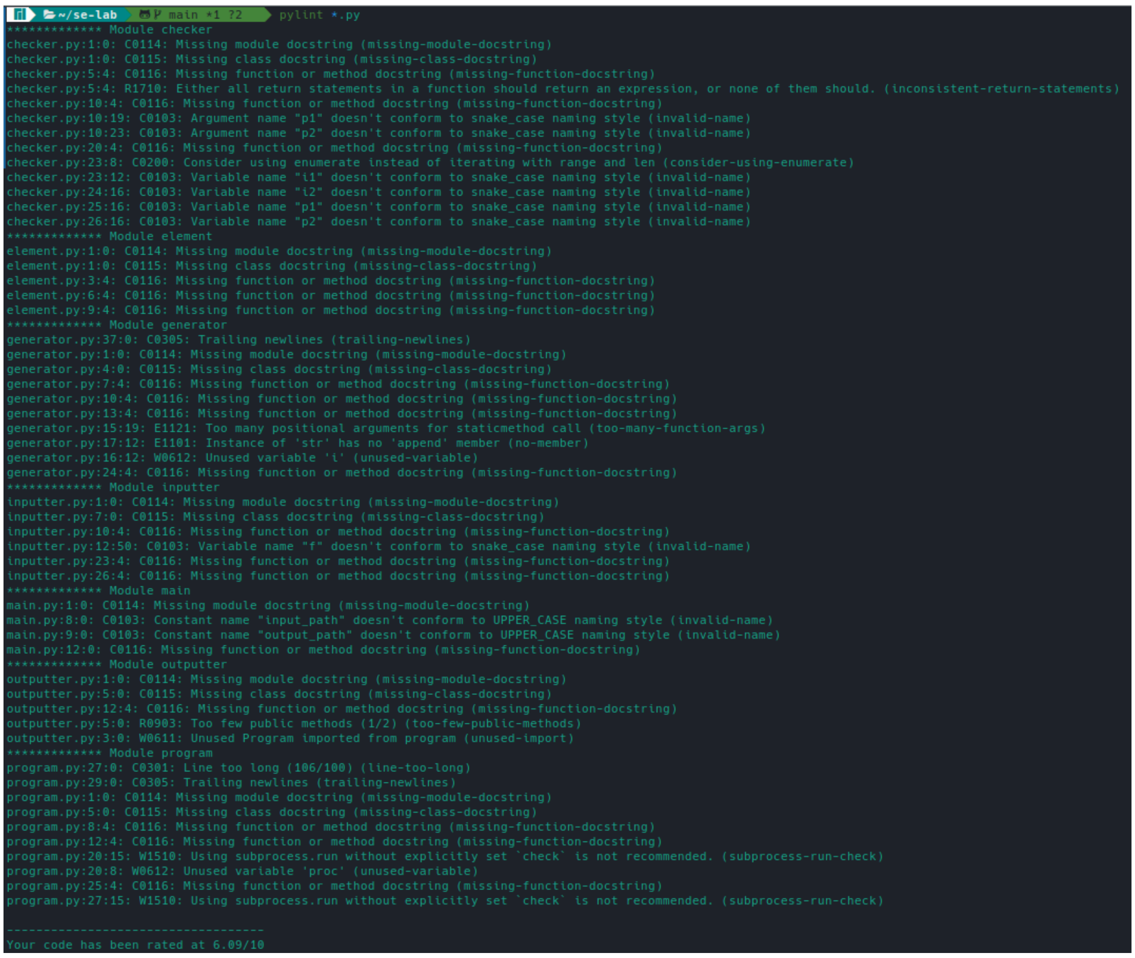
1. 选取pylint, 下载安装pylint:



```
~ pip install pylint
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: pylint in ./.local/lib/python3.10/site-packages (2.15.5)
Requirement already satisfied: tomli>=1.1.0 in /usr/lib/python3.10/site-packages (from pylint) (2.0.1)
Requirement already satisfied: mccabe<0.8,>=0.6 in ./.local/lib/python3.10/site-packages (from pylint) (0.7.0)
Requirement already satisfied: astroid<=2.14.0-dev0,>=2.12.12 in ./.local/lib/python3.10/site-packages (from pylint)
 (2.12.12)
Requirement already satisfied: platformdirs>=2.2.0 in /usr/lib/python3.10/site-packages (from pylint) (2.5.3)
Requirement already satisfied: dill>=0.2 in ./.local/lib/python3.10/site-packages (from pylint) (0.3.6)
Requirement already satisfied: tomlkit>=0.10.1 in ./.local/lib/python3.10/site-packages (from pylint) (0.11.6)
Requirement already satisfied: isort<6,>=4.2.5 in ./.local/lib/python3.10/site-packages (from pylint) (5.10.1)
Requirement already satisfied: wrapt<2,>=1.11 in ./.local/lib/python3.10/site-packages (from astroid<=2.14.0-dev0,>=
2.12.12->pylint) (1.14.1)
Requirement already satisfied: lazy-object-proxy>=1.4.0 in ./.local/lib/python3.10/site-packages (from astroid<=2.14
.0-dev0,>=2.12.12->pylint) (1.8.0)
~
```

2. 使用说明：执行 `pylint [filename]` 进行检测。
   pylint 会显示出每个模块中的静态检查结果，其中，C代表Convention, E代表Error, R代表Refactor, W代表Warning，在这后面还有详细的信息提示
   最后pylint会为代码评分。

3. 在 `se-lab` 目录下执行 `pylint *.py` 进行检测。
   得到下图结果：

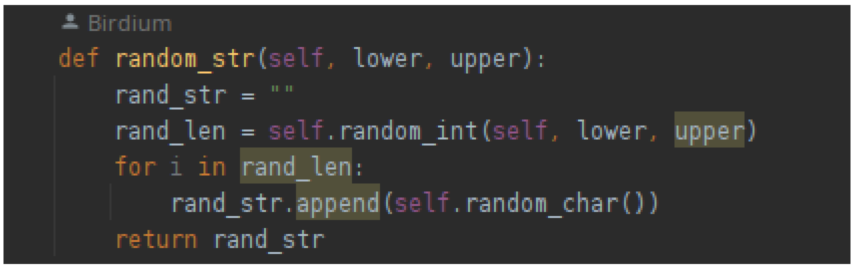


```
~/se-lab  main *1 ?2  pylint *.py
************* Module checker
checker.py:1:0: C0114: Missing module docstring (missing-module-docstring)
checker.py:1:0: C0115: Missing class docstring (missing-class-docstring)
checker.py:5:4: C0116: Missing function or method docstring (missing-function-docstring)
checker.py:5:4: R1710: Either all return statements in a function should return an expression, or none of them should. (inconsistent-return-statements)
checker.py:10:4: C0116: Missing function or method docstring (missing-function-docstring)
checker.py:10:19: C0103: Argument name "p1" doesn't conform to snake_case naming style (invalid-name)
checker.py:10:23: C0103: Argument name "p2" doesn't conform to snake_case naming style (invalid-name)
checker.py:20:4: C0116: Missing function or method docstring (missing-function-docstring)
checker.py:23:8: C0200: Consider using enumerate instead of iterating with range and len (consider-using-enumerate)
checker.py:23:12: C0103: Variable name "i1" doesn't conform to snake_case naming style (invalid-name)
checker.py:24:16: C0103: Variable name "i2" doesn't conform to snake_case naming style (invalid-name)
checker.py:25:16: C0103: Variable name "p1" doesn't conform to snake_case naming style (invalid-name)
checker.py:26:16: C0103: Variable name "p2" doesn't conform to snake_case naming style (invalid-name)
************* Module element
element.py:1:0: C0114: Missing module docstring (missing-module-docstring)
element.py:1:0: C0115: Missing class docstring (missing-class-docstring)
element.py:3:4: C0116: Missing function or method docstring (missing-function-docstring)
element.py:6:4: C0116: Missing function or method docstring (missing-function-docstring)
element.py:9:4: C0116: Missing function or method docstring (missing-function-docstring)
************* Module generator
generator.py:37:0: C0305: Trailing newlines (trailing-newlines)
generator.py:1:0: C0114: Missing module docstring (missing-module-docstring)
generator.py:4:0: C0115: Missing class docstring (missing-class-docstring)
generator.py:7:4: C0116: Missing function or method docstring (missing-function-docstring)
generator.py:10:4: C0116: Missing function or method docstring (missing-function-docstring)
generator.py:13:4: C0116: Missing function or method docstring (missing-function-docstring)
generator.py:15:19: E1121: Too many positional arguments for staticmethod call (too-many-function-args)
generator.py:17:12: E1101: Instance of 'str' has no 'append' member (no-member)
generator.py:16:12: W0612: Unused variable 'i' (unused-variable)
generator.py:24:4: C0116: Missing function or method docstring (missing-function-docstring)
************* Module inputter
inputter.py:1:0: C0114: Missing module docstring (missing-module-docstring)
inputter.py:7:0: C0115: Missing class docstring (missing-class-docstring)
inputter.py:10:4: C0116: Missing function or method docstring (missing-function-docstring)
inputter.py:12:50: C0103: Variable name "f" doesn't conform to snake_case naming style (invalid-name)
inputter.py:23:4: C0116: Missing function or method docstring (missing-function-docstring)
inputter.py:26:4: C0116: Missing function or method docstring (missing-function-docstring)
************* Module main
main.py:1:0: C0114: Missing module docstring (missing-module-docstring)
main.py:8:0: C0103: Constant name "input_path" doesn't conform to UPPER_CASE naming style (invalid-name)
main.py:9:0: C0103: Constant name "output_path" doesn't conform to UPPER_CASE naming style (invalid-name)
main.py:12:0: C0116: Missing function or method docstring (missing-function-docstring)
************* Module outputter
outputter.py:1:0: C0114: Missing module docstring (missing-module-docstring)
outputter.py:5:0: C0115: Missing class docstring (missing-class-docstring)
outputter.py:12:4: C0116: Missing function or method docstring (missing-function-docstring)
outputter.py:5:0: R0903: Too few public methods (1/2) (too-few-public-methods)
outputter.py:3:0: W0611: Unused Program imported from program (unused-import)
************* Module program
program.py:27:0: C0301: Line too long (106/100) (line-too-long)
program.py:29:0: C0305: Trailing newlines (trailing-newlines)
program.py:1:0: C0114: Missing module docstring (missing-module-docstring)
program.py:5:0: C0115: Missing class docstring (missing-class-docstring)
program.py:8:4: C0116: Missing function or method docstring (missing-function-docstring)
program.py:12:4: C0116: Missing function or method docstring (missing-function-docstring)
program.py:20:15: W1510: Using subprocess.run without explicitly set `check` is not recommended. (subprocess-run-check)
program.py:20:8: W0612: Unused variable 'proc' (unused-variable)
program.py:25:4: C0116: Missing function or method docstring (missing-function-docstring)
program.py:27:15: W1510: Using subprocess.run without explicitly set `check` is not recommended. (subprocess-run-check)

-----------------------------------
Your code has been rated at 6.09/10
```

   可以看到代码是非常烂的。。甚至还有Error

4. 接下来对代码进行修复：

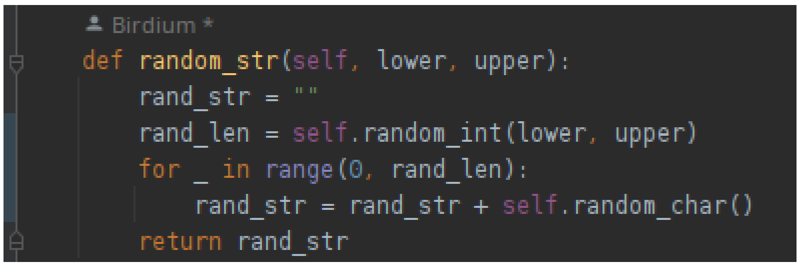
1. 先从有错误的Module generator修起：



```
Birdium
def random_str(self, lower, upper):
    rand_str = ""
    rand_len = self.random_int(self, lower, upper)
    for i in rand_len:
        rand_str.append(self.random_char())
    return rand_str
```

我们看到这个小小的函数中竟然有三个错误！一个是参数过多，一个是 for in 语句的错误，还有一个是调用了str实例不存在的方法。但是奇怪的是，这个函数理应是会被频繁使用的，我去看了下lab4的两个测试样例，果然这两个样例没有生成任何string，因此我的程序完全可以运行测试样例而不报错。这验证了“未测试代码永远是错的”这一观点，由此可以说明测试是很重要的。

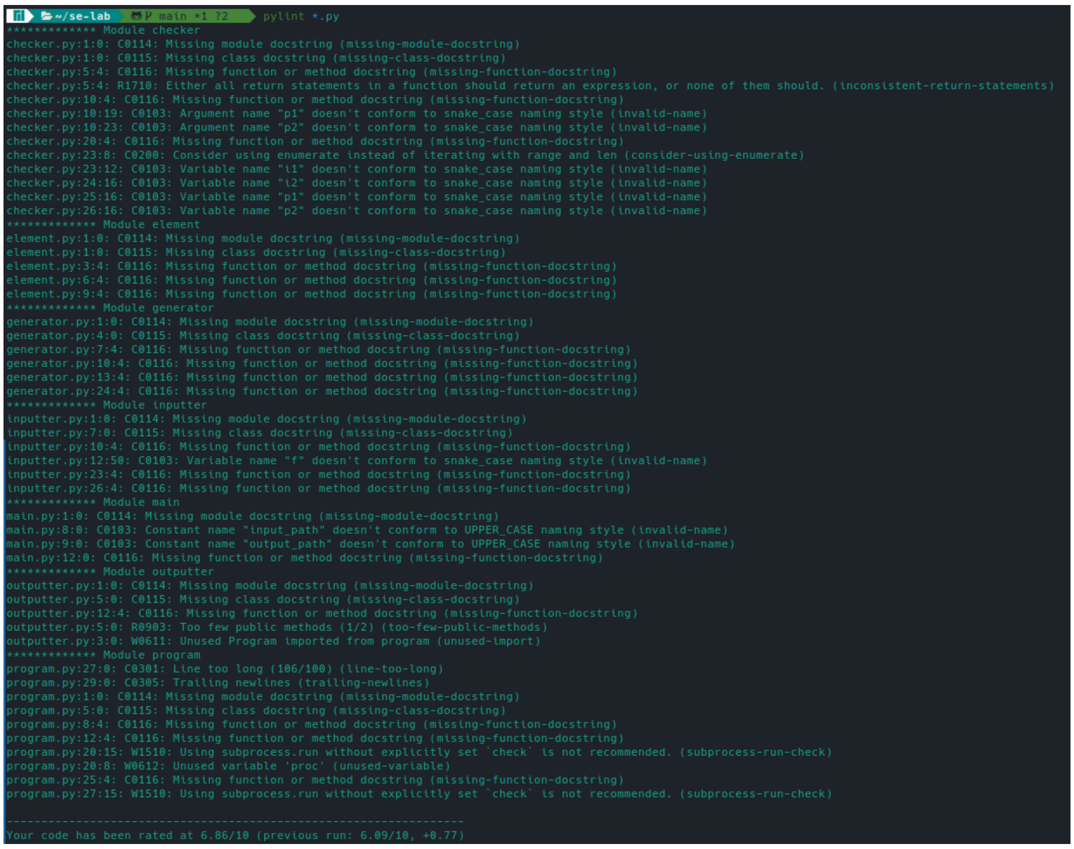
修复后的代码：



```
Birdium *
def random_str(self, lower, upper):
    rand_str = ""
    rand_len = self.random_int(lower, upper)
    for _ in range(0, rand_len):
        rand_str = rand_str + self.random_char()
    return rand_str
```

修复了三个错误，同时还修复了一个Warning: Unused variable 'i', 我使用了python的一个特性'_', 可以避免这一操作。

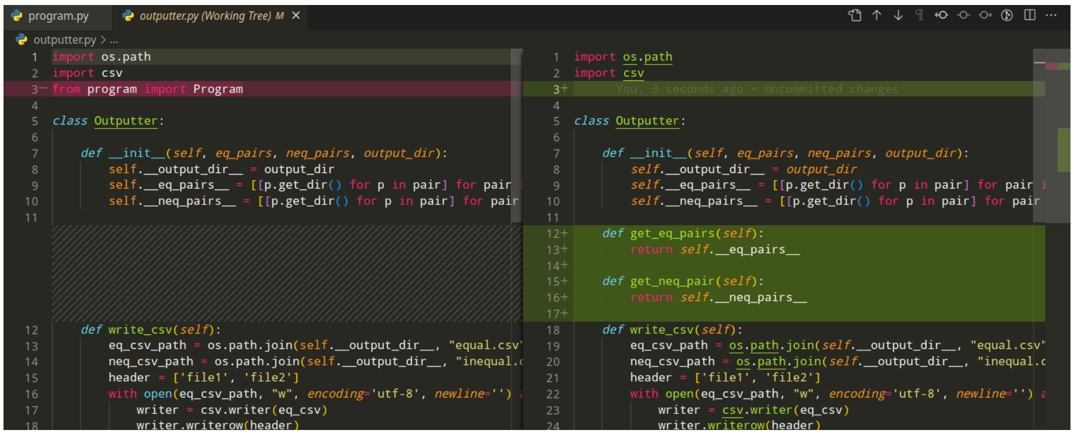
修复后重新跑分结果：



```
~/se-lab  main *1 ?2  pylint *.py
************* Module checker
checker.py:1:0: C0114: Missing module docstring (missing-module-docstring)
checker.py:1:0: C0115: Missing class docstring (missing-class-docstring)
checker.py:5:4: C0116: Missing function or method docstring (missing-function-docstring)
checker.py:5:4: R1710: Either all return statements in a function should return an expression, or none of them should. (inconsistent-return-statements)
checker.py:10:4: C0116: Missing function or method docstring (missing-function-docstring)
checker.py:10:19: C0103: Argument name "p1" doesn't conform to snake_case naming style (invalid-name)
checker.py:10:23: C0103: Argument name "p2" doesn't conform to snake_case naming style (invalid-name)
checker.py:20:4: C0116: Missing function or method docstring (missing-function-docstring)
checker.py:23:8: C0200: Consider using enumerate instead of iterating with range and len (consider-using-enumerate)
checker.py:23:12: C0103: Variable name "i1" doesn't conform to snake_case naming style (invalid-name)
checker.py:24:16: C0103: Variable name "i2" doesn't conform to snake_case naming style (invalid-name)
checker.py:25:16: C0103: Variable name "p1" doesn't conform to snake_case naming style (invalid-name)
checker.py:26:16: C0103: Variable name "p2" doesn't conform to snake_case naming style (invalid-name)
************* Module element
element.py:1:0: C0114: Missing module docstring (missing-module-docstring)
element.py:1:0: C0115: Missing class docstring (missing-class-docstring)
element.py:3:4: C0116: Missing function or method docstring (missing-function-docstring)
element.py:6:4: C0116: Missing function or method docstring (missing-function-docstring)
element.py:9:4: C0116: Missing function or method docstring (missing-function-docstring)
************* Module generator
generator.py:1:0: C0114: Missing module docstring (missing-module-docstring)
generator.py:4:0: C0115: Missing class docstring (missing-class-docstring)
generator.py:7:4: C0116: Missing function or method docstring (missing-function-docstring)
generator.py:10:4: C0116: Missing function or method docstring (missing-function-docstring)
generator.py:13:4: C0116: Missing function or method docstring (missing-function-docstring)
generator.py:24:4: C0116: Missing function or method docstring (missing-function-docstring)
************* Module inputter
inputter.py:1:0: C0114: Missing module docstring (missing-module-docstring)
inputter.py:7:0: C0115: Missing class docstring (missing-class-docstring)
inputter.py:10:4: C0116: Missing function or method docstring (missing-function-docstring)
inputter.py:12:50: C0103: Variable name "f" doesn't conform to snake_case naming style (invalid-name)
inputter.py:23:4: C0116: Missing function or method docstring (missing-function-docstring)
inputter.py:26:4: C0116: Missing function or method docstring (missing-function-docstring)
************* Module main
main.py:1:0: C0114: Missing module docstring (missing-module-docstring)
main.py:8:0: C0103: Constant name "input_path" doesn't conform to UPPER_CASE naming style (invalid-name)
main.py:9:0: C0103: Constant name "output_path" doesn't conform to UPPER_CASE naming style (invalid-name)
main.py:12:0: C0116: Missing function or method docstring (missing-function-docstring)
************* Module outputter
outputter.py:1:0: C0114: Missing module docstring (missing-module-docstring)
outputter.py:5:0: C0115: Missing class docstring (missing-class-docstring)
outputter.py:12:4: C0116: Missing function or method docstring (missing-function-docstring)
outputter.py:5:0: R0903: Too few public methods (1/2) (too-few-public-methods)
outputter.py:3:0: W0611: Unused Program imported from program (unused-import)
************* Module program
program.py:27:0: C0301: Line too long (106/100) (line-too-long)
program.py:29:0: C0305: Trailing newlines (trailing-newlines)
program.py:1:0: C0114: Missing module docstring (missing-module-docstring)
program.py:5:0: C0115: Missing class docstring (missing-class-docstring)
program.py:8:4: C0116: Missing function or method docstring (missing-function-docstring)
program.py:12:4: C0116: Missing function or method docstring (missing-function-docstring)
program.py:20:15: W1510: Using subprocess.run without explicitly set `check` is not recommended. (subprocess-run-check)
program.py:20:8: W0612: Unused variable 'proc' (unused-variable)
program.py:25:4: C0116: Missing function or method docstring (missing-function-docstring)
program.py:27:15: W1510: Using subprocess.run without explicitly set `check` is not recommended. (subprocess-run-check)

------------------------------------------------------------------
Your code has been rated at 6.86/10 (previous run: 6.09/10, +0.77)
```

可以看到，此前generator的E, W提示消失了。

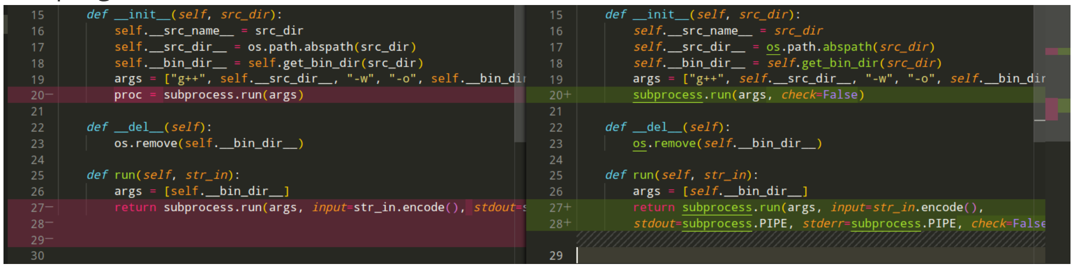
2. 修复outputter:



```python
import os.path
import csv
from program import Program

class Outputter:

    def __init__(self, eq_pairs, neq_pairs, output_dir):
        self.__output_dir__ = output_dir
        self.__eq_pairs__ = [[p.get_dir() for p in pair] for pair
        self.__neq_pairs__ = [[p.get_dir() for p in pair] for pair

    def write_csv(self):
        eq_csv_path = os.path.join(self.__output_dir__, "equal.csv'
        neq_csv_path = os.path.join(self.__output_dir__, "inequal.c
        header = ['file1', 'file2']
        with open(eq_csv_path, "w", encoding='utf-8', newline='')
            writer = csv.writer(eq_csv)
            writer.writerow(header)
```

```python
import os.path
import csv
      You, 3 seconds ago • Uncommitted changes

class Outputter:

    def __init__(self, eq_pairs, neq_pairs, output_dir):
        self.__output_dir__ = output_dir
        self.__eq_pairs__ = [[p.get_dir() for p in pair] for pair
        self.__neq_pairs__ = [[p.get_dir() for p in pair] for pair

    def get_eq_pairs(self):
        return self.__eq_pairs__

    def get_neq_pair(self):
        return self.__neq_pairs__

    def write_csv(self):
        eq_csv_path = os.path.join(self.__output_dir__, "equal.csv"
        neq_csv_path = os.path.join(self.__output_dir__, "inequal.c
        header = ['file1', 'file2']
        with open(eq_csv_path, "w", encoding='utf-8', newline='')
            writer = csv.writer(eq_csv)
            writer.writerow(header)
```

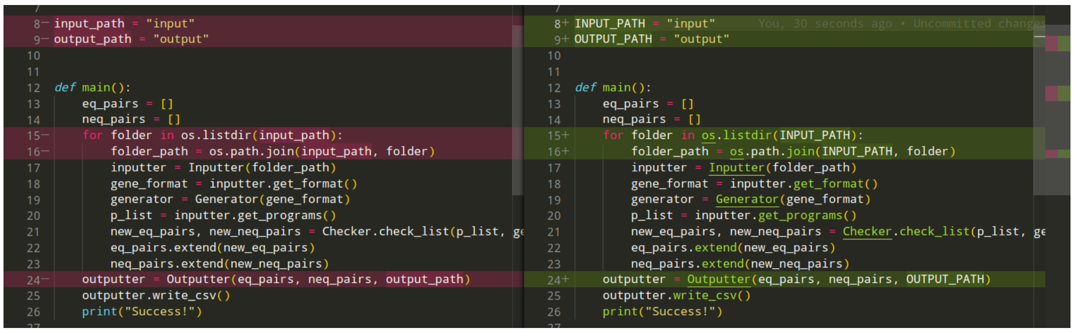
3. 修复program:



```python
    def __init__(self, src_dir):
        self.__src_name__ = src_dir
        self.__src_dir__ = os.path.abspath(src_dir)
        self.__bin_dir__ = self.get_bin_dir(src_dir)
        args = ["g++", self.__src_dir__, "-w", "-o", self.__bin_di
        proc = subprocess.run(args)

    def __del__(self):
        os.remove(self.__bin_dir__)

    def run(self, str_in):
        args = [self.__bin_dir__]
        return subprocess.run(args, input=str_in.encode(), stdout=s


```

```python
    def __init__(self, src_dir):
        self.__src_name__ = src_dir
        self.__src_dir__ = os.path.abspath(src_dir)
        self.__bin_dir__ = self.get_bin_dir(src_dir)
        args = ["g++", self.__src_dir__, "-w", "-o", self.__bin_dir
        subprocess.run(args, check=False)

    def __del__(self):
        os.remove(self.__bin_dir__)

    def run(self, str_in):
        args = [self.__bin_dir__]
        return subprocess.run(args, input=str_in.encode(),
        stdout=subprocess.PIPE, stderr=subprocess.PIPE, check=False

```

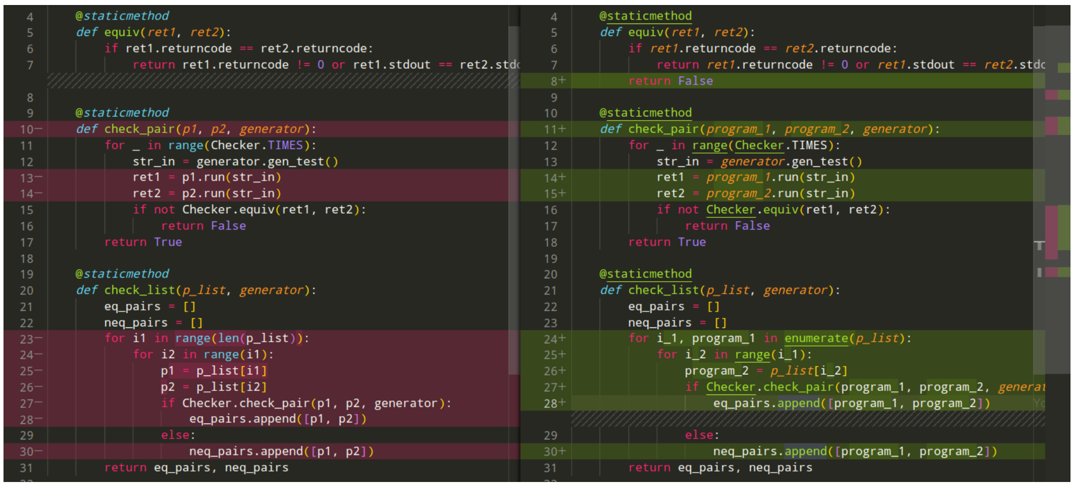
4. 修复main:



```python
input_path = "input"
output_path = "output"


def main():
    eq_pairs = []
    neq_pairs = []
    for folder in os.listdir(input_path):
        folder_path = os.path.join(input_path, folder)
        inputter = Inputter(folder_path)
        gene_format = inputter.get_format()
        generator = Generator(gene_format)
        p_list = inputter.get_programs()
        new_eq_pairs, new_neq_pairs = Checker.check_list(p_list, ge
        eq_pairs.extend(new_eq_pairs)
        neq_pairs.extend(new_neq_pairs)
    outputter = Outputter(eq_pairs, neq_pairs, output_path)
    outputter.write_csv()
    print("Success!")
```

```python
INPUT_PATH = "input"      You, 30 seconds ago • Uncommitted change
OUTPUT_PATH = "output"


def main():
    eq_pairs = []
    neq_pairs = []
    for folder in os.listdir(INPUT_PATH):
        folder_path = os.path.join(INPUT_PATH, folder)
        inputter = Inputter(folder_path)
        gene_format = inputter.get_format()
        generator = Generator(gene_format)
        p_list = inputter.get_programs()
        new_eq_pairs, new_neq_pairs = Checker.check_list(p_list, ge
        eq_pairs.extend(new_eq_pairs)
        neq_pairs.extend(new_neq_pairs)
    outputter = Outputter(eq_pairs, neq_pairs, OUTPUT_PATH)
    outputter.write_csv()
    print("Success!")
```

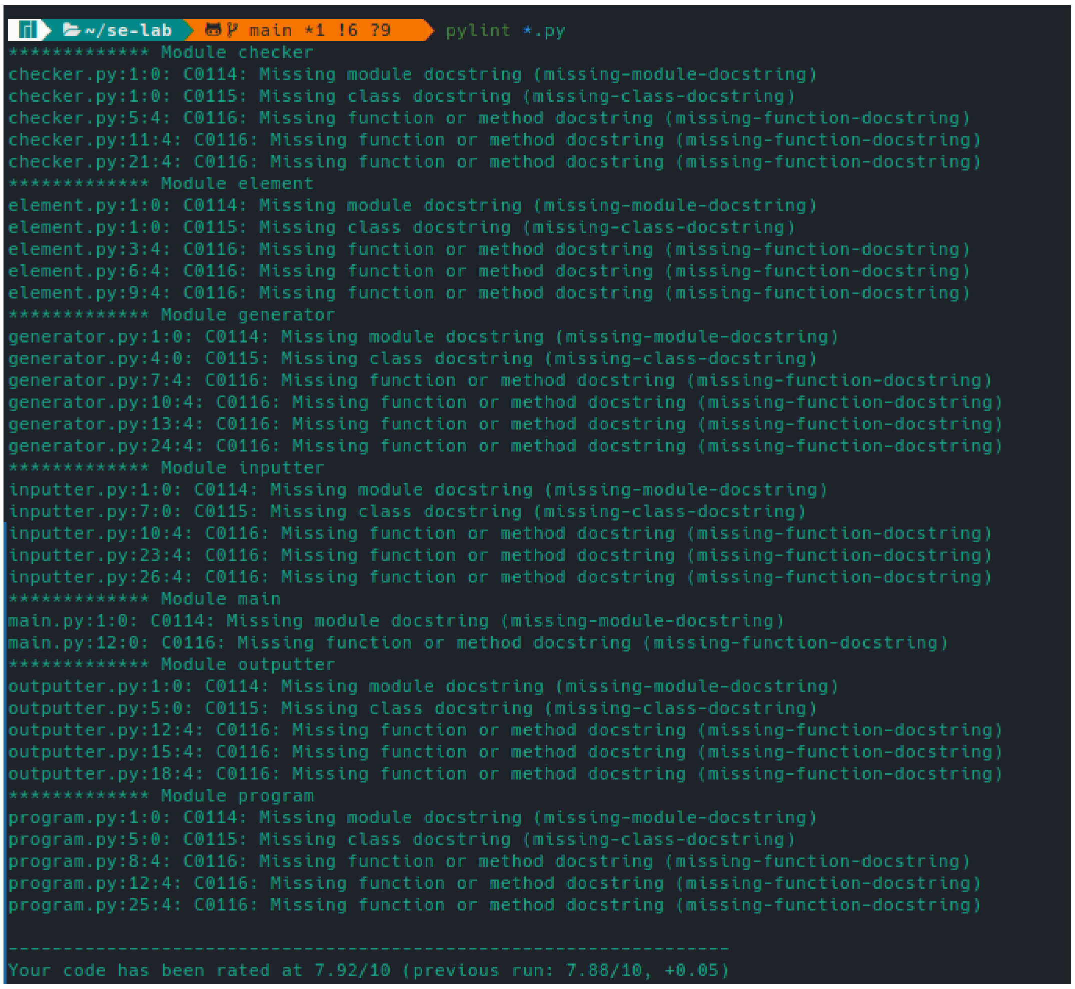
5. 修复checker:



```python
    @staticmethod
    def equiv(ret1, ret2):
        if ret1.returncode == ret2.returncode:
            return ret1.returncode != 0 or ret1.stdout == ret2.stdc

    @staticmethod
    def check_pair(p1, p2, generator):
        for _ in range(Checker.TIMES):
            str_in = generator.gen_test()
            ret1 = p1.run(str_in)
            ret2 = p2.run(str_in)
            if not Checker.equiv(ret1, ret2):
                return False
        return True

    @staticmethod
    def check_list(p_list, generator):
        eq_pairs = []
        neq_pairs = []
        for i1 in range(len(p_list)):
            for i2 in range(i1):
                p1 = p_list[i1]
                p2 = p_list[i2]
                if Checker.check_pair(p1, p2, generator):
                    eq_pairs.append([p1, p2])
                else:
                    neq_pairs.append([p1, p2])
        return eq_pairs, neq_pairs
```

```python
    @staticmethod
    def equiv(ret1, ret2):
        if ret1.returncode == ret2.returncode:
            return ret1.returncode != 0 or ret1.stdout == ret2.stdc
        return False

    @staticmethod
    def check_pair(program_1, program_2, generator):
        for _ in range(Checker.TIMES):
            str_in = generator.gen_test()
            ret1 = program_1.run(str_in)
            ret2 = program_2.run(str_in)
            if not Checker.equiv(ret1, ret2):
                return False
        return True

    @staticmethod
    def check_list(p_list, generator):
        eq_pairs = []
        neq_pairs = []
        for i_1, program_1 in enumerate(p_list):
            for i_2 in range(i_1):
                program_2 = p_list[i_2]
                if Checker.check_pair(program_1, program_2, generat
                    eq_pairs.append([program_1, program_2])
                else:
                    neq_pairs.append([program_1, program_2])
        return eq_pairs, neq_pairs
```

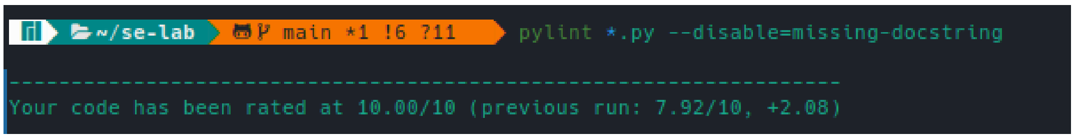
6. 修复完大多数内容之后的结果：



```
 ~/se-lab   main *1 !6 ?9   pylint *.py
************* Module checker
checker.py:1:0: C0114: Missing module docstring (missing-module-docstring)
checker.py:1:0: C0115: Missing class docstring (missing-class-docstring)
checker.py:5:4: C0116: Missing function or method docstring (missing-function-docstring)
checker.py:11:4: C0116: Missing function or method docstring (missing-function-docstring)
checker.py:21:4: C0116: Missing function or method docstring (missing-function-docstring)
************* Module element
element.py:1:0: C0114: Missing module docstring (missing-module-docstring)
element.py:1:0: C0115: Missing class docstring (missing-class-docstring)
element.py:3:4: C0116: Missing function or method docstring (missing-function-docstring)
element.py:6:4: C0116: Missing function or method docstring (missing-function-docstring)
element.py:9:4: C0116: Missing function or method docstring (missing-function-docstring)
************* Module generator
generator.py:1:0: C0114: Missing module docstring (missing-module-docstring)
generator.py:4:0: C0115: Missing class docstring (missing-class-docstring)
generator.py:7:4: C0116: Missing function or method docstring (missing-function-docstring)
generator.py:10:4: C0116: Missing function or method docstring (missing-function-docstring)
generator.py:13:4: C0116: Missing function or method docstring (missing-function-docstring)
generator.py:24:4: C0116: Missing function or method docstring (missing-function-docstring)
************* Module inputter
inputter.py:1:0: C0114: Missing module docstring (missing-module-docstring)
inputter.py:7:0: C0115: Missing class docstring (missing-class-docstring)
inputter.py:10:4: C0116: Missing function or method docstring (missing-function-docstring)
inputter.py:23:4: C0116: Missing function or method docstring (missing-function-docstring)
inputter.py:26:4: C0116: Missing function or method docstring (missing-function-docstring)
************* Module main
main.py:1:0: C0114: Missing module docstring (missing-module-docstring)
main.py:12:0: C0116: Missing function or method docstring (missing-function-docstring)
************* Module outputter
outputter.py:1:0: C0114: Missing module docstring (missing-module-docstring)
outputter.py:5:0: C0115: Missing class docstring (missing-class-docstring)
outputter.py:12:4: C0116: Missing function or method docstring (missing-function-docstring)
outputter.py:15:4: C0116: Missing function or method docstring (missing-function-docstring)
outputter.py:18:4: C0116: Missing function or method docstring (missing-function-docstring)
************* Module program
program.py:1:0: C0114: Missing module docstring (missing-module-docstring)
program.py:5:0: C0115: Missing class docstring (missing-class-docstring)
program.py:8:4: C0116: Missing function or method docstring (missing-function-docstring)
program.py:12:4: C0116: Missing function or method docstring (missing-function-docstring)
program.py:25:4: C0116: Missing function or method docstring (missing-function-docstring)

------------------------------------------------------------------
Your code has been rated at 7.92/10 (previous run: 7.88/10, +0.05)
```

消除了除了docstring以外的所有提示

选择忽略docstring：执行 `pylint *.py --disable=missing-docstring`，得到了Pylint的满分结果：

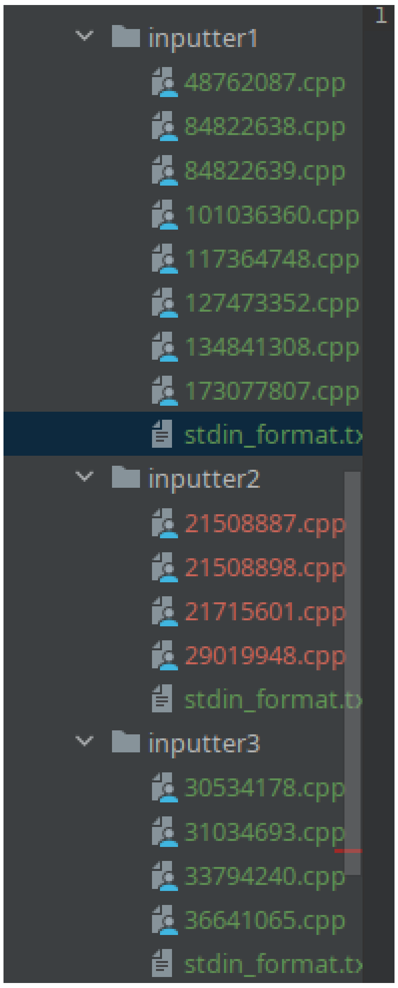


```
 ~/se-lab   main *1 !6 ?11   pylint *.py --disable=missing-docstring

-------------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 7.92/10, +2.08)
```

二、单元测试

1. 测试目的：发现程序中的错误
   测试对象：等价判断工具中的所有程序
   测试环境：Manjaro Linux, python 3.10.8, PyCharm
   测试工具：python自带的unittest, Pycharm, coverage

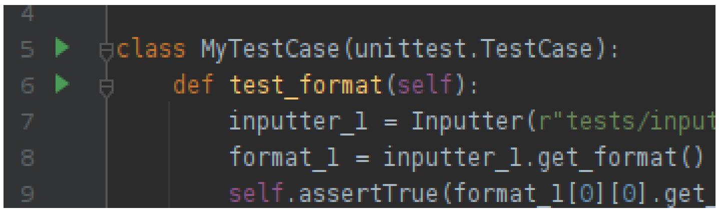
2. 首先我们对input部分进行单元测试：
   测试目的：测试input的实现中是否有错误
   测试用例：如下文件：



   测试使用的代码：包含了测试用例和预期输出

```python
class MyTestCase(unittest.TestCase):
    def test_format(self):
        inputter_1 = Inputter(r"tests/inputter1")
        format_1 = inputter_1.get_format()
        self.assertTrue(format_1[0][0].get_type() == 'int')
        self.assertTrue(format_1[0][0].get_lower() == 1)
        self.assertTrue(format_1[0][0].get_upper() == 2)
        self.assertTrue(format_1[0][1].get_type() == 'string')
        self.assertTrue(format_1[0][1].get_lower() == 2)
        self.assertTrue(format_1[0][1].get_upper() == 3)

        inputter_2 = Inputter(r"tests/inputter2")
        format_2 = inputter_2.get_format()
        self.assertTrue(format_2[0][0].get_type() == 'int')
        self.assertTrue(format_2[0][0].get_lower() == 114)
        self.assertTrue(format_2[0][0].get_upper() == 514)
        self.assertTrue(format_2[0][1].get_type() == 'int')
        self.assertTrue(format_2[0][1].get_lower() == 191)
        self.assertTrue(format_2[0][1].get_upper() == 9810)

        inputter_3 = Inputter(r"tests/inputter3")
        format_3 = inputter_3.get_format()
        self.assertTrue(format_3[0][0].get_type() == 'int')
        self.assertTrue(format_3[0][0].get_lower() == 114)
        self.assertTrue(format_3[0][0].get_upper() == 514)
        self.assertTrue(format_3[0][1].get_type() == 'int')
        self.assertTrue(format_3[0][1].get_lower() == 191)
        self.assertTrue(format_3[0][1].get_upper() == 9810)
        self.assertTrue(format_3[1][0].get_type() == 'char')
        self.assertTrue(format_3[1][1].get_type() == 'char')
        self.assertTrue(format_3[1][2].get_type() == 'char')
        self.assertTrue(format_3[1][3].get_type() == 'char')
        self.assertTrue(format_3[1][4].get_type() == 'char')
        self.assertTrue(format_3[2][0].get_type() == 'string')
        self.assertTrue(format_3[2][0].get_lower() == 19)
        self.assertTrue(format_3[2][0].get_upper() == 19)

    def test_program(self):
        inputter_1 = Inputter(r"tests/inputter1")
        program_1 = inputter_1.get_programs()
        prog_dir_1 = [prog.get_dir() for prog in program_1]
        std_dir_1 = ['tests/inputter1/48762087.cpp',
'tests/inputter1/84822638.cpp', 'tests/inputter1/84822639.cpp',
                     'tests/inputter1/101036360.cpp',
'tests/inputter1/117364748.cpp', 'tests/inputter1/127473352.cpp',
                     'tests/inputter1/134841308.cpp',
'tests/inputter1/173077807.cpp']
        self.assertEqual(len(prog_dir_1), len(std_dir_1))
        self.assertTrue(len(set(prog_dir_1).difference(set(std_dir_1))) == 0)

        inputter_2 = Inputter(r"tests/inputter2")
        program_2 = inputter_2.get_programs()
        prog_dir_2 = [prog.get_dir() for prog in program_2]
        std_dir_2 = ['tests/inputter2/21508887.cpp',
'tests/inputter2/21508898.cpp', 'tests/inputter2/21715601.cpp',
                     'tests/inputter2/29019948.cpp']
        self.assertEqual(len(prog_dir_2), len(std_dir_2))
        self.assertTrue(len(set(prog_dir_2).difference(set(std_dir_2))) == 0)

        inputter_3 = Inputter(r"tests/inputter3")
        program_3 = inputter_3.get_programs()
        prog_dir_3 = [prog.get_dir() for prog in program_3]
        std_dir_3 = ['tests/inputter3/30534178.cpp',
'tests/inputter3/31034693.cpp', 'tests/inputter3/33794240.cpp',
                     'tests/inputter3/36641065.cpp']
        self.assertEqual(len(prog_dir_3), len(std_dir_3))
        self.assertTrue(len(set(prog_dir_3).difference(set(std_dir_3))) == 0)


if __name__ == '__main__':
    unittest.main()
```

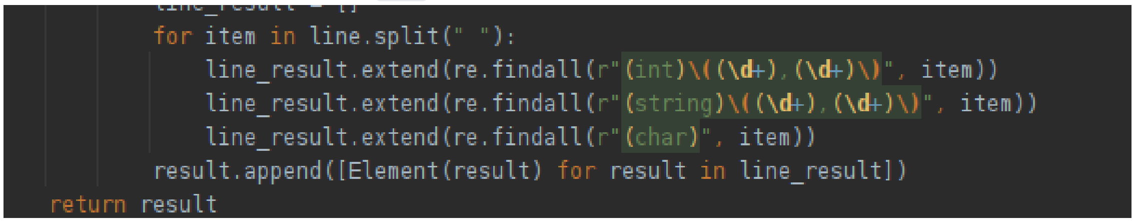
测试方法：使用PyCharm IDE自带的测试功能:



```
class MyTestCase(unittest.TestCase):
    def test_format(self):
        inputter_1 = Inputter(r"tests/input
        format_1 = inputter_1.get_format()
        self.assertTrue(format_1[0][0].get_
```

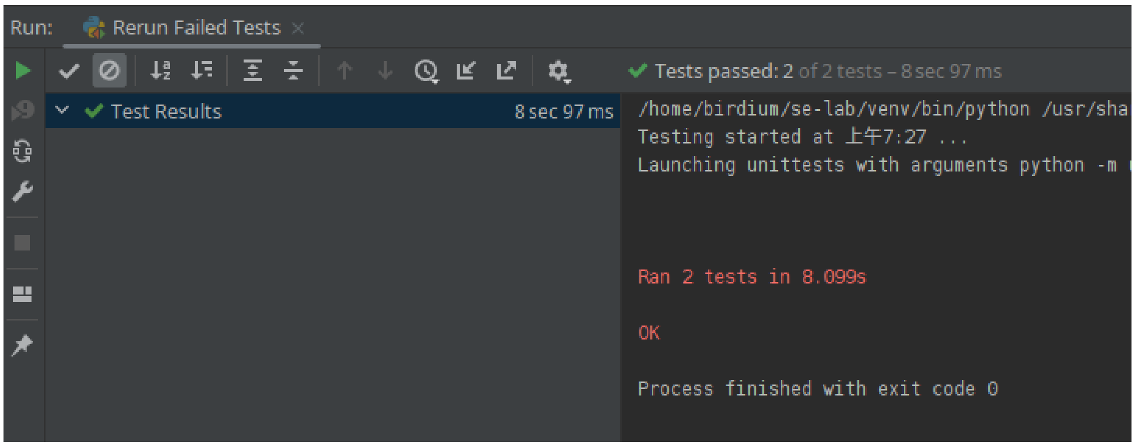
运行右边的绿色小按钮来进行测试

实际输出：出现了一些bug，对于 char 类型的输入，判断输出中对应行列的`Element.get_type() == "char"`, 但是实际上却输出了 `"c"`。检查的时候发现如下图所示的正则表达式中，上面两条捕获的类型是元组，但是char对应的捕获结果只是一个字符串。因此get_type获取`item[0]` 时，实际上获取了字符串的第一个元素 `"c"`。



```
            line_result = []
            for item in line.split(" "):
                line_result.extend(re.findall(r"(int)\(\(\d+),(\d+)\)", item))
                line_result.extend(re.findall(r"(string)\(\(\d+),(\d+)\)", item))
                line_result.extend(re.findall(r"(char)", item))
            result.append([Element(result) for result in line_result])
    return result
```

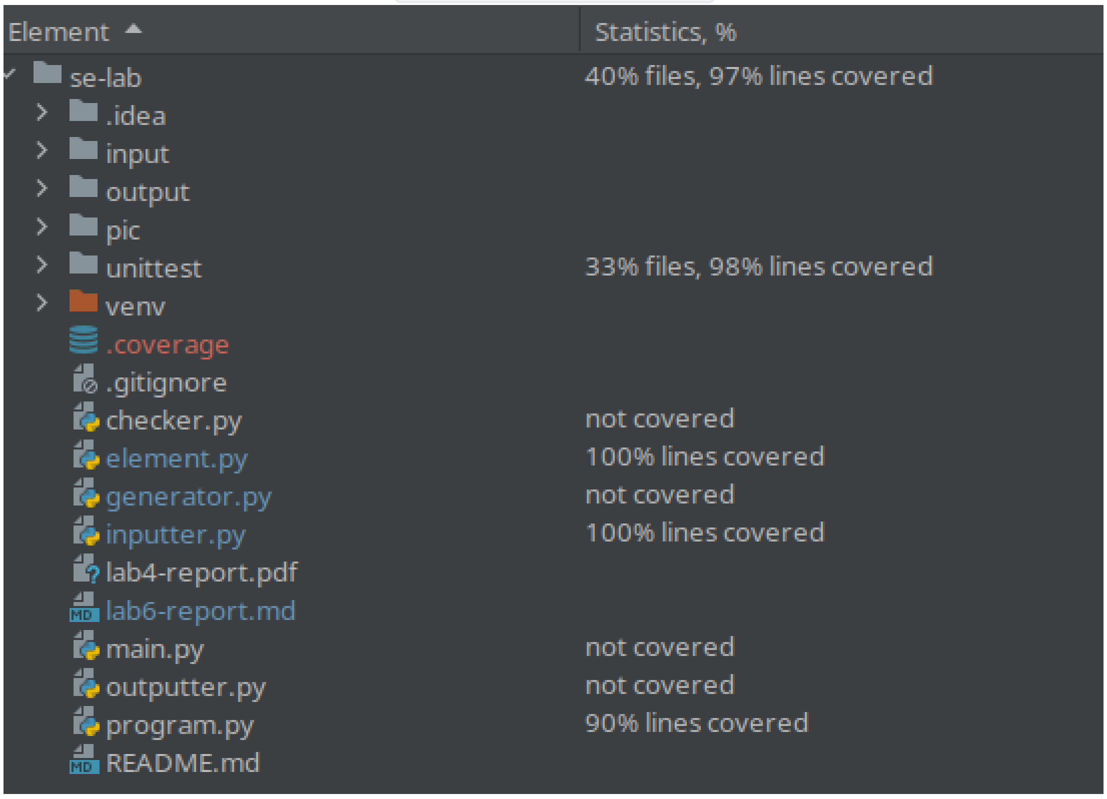
修复完之后通过了所有测试：



```
/home/birdium/se-lab/venv/bin/python /usr/sha
Testing started at 上午7:27 ...
Launching unittests with arguments python -m 


Ran 2 tests in 8.099s

OK

Process finished with exit code 0
```

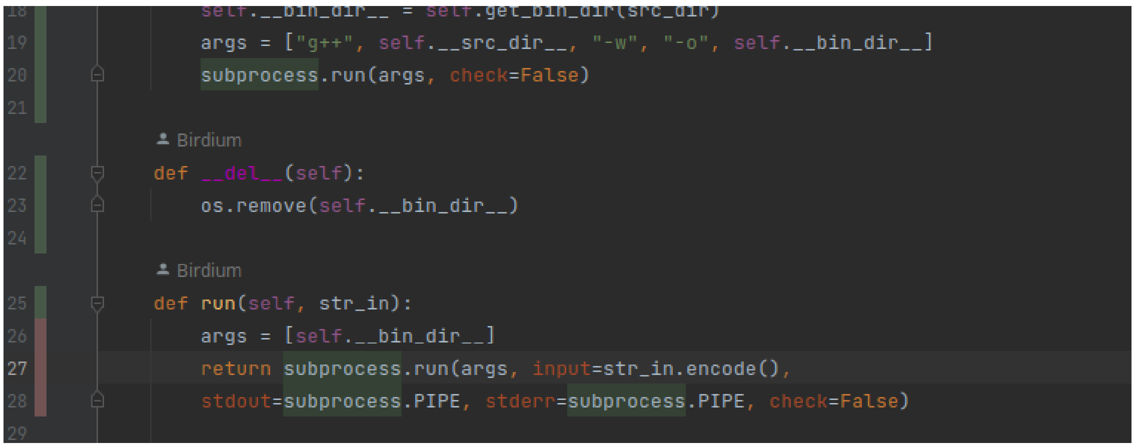
单元测试覆盖率：覆盖了`input.py` 中的全部代码和 `element.py` 中全部代码。

下面我们使用coverage工具来检查覆盖率：PyCharm Professional Edition中自带了这一功能：我们只需要点击绿色小三角，并选择`test ... with coverage`，就能获得Coverage报告：



Element	Statistics, %
se-lab	40% files, 97% lines covered
.idea	
input	
output	
pic	
unittest	33% files, 98% lines covered
venv	
.coverage	
.gitignore	
checker.py	not covered
element.py	100% lines covered
generator.py	not covered
inputter.py	100% lines covered
lab4-report.pdf	
lab6-report.md	
main.py	not covered
outputter.py	not covered
program.py	90% lines covered
README.md	

我们可以看到，不仅覆盖了input 和 element，而且还覆盖了program中除了 `run()` 方法以外的方法。



```
        self.__bin_dir__ = self.get_bin_dir(src_dir)
        args = ["g++", self.__src_dir__, "-w", "-o", self.__bin_dir__]
        subprocess.run(args, check=False)

    def __del__(self):
        os.remove(self.__bin_dir__)

    def run(self, str_in):
        args = [self.__bin_dir__]
        return subprocess.run(args, input=str_in.encode(),
        stdout=subprocess.PIPE, stderr=subprocess.PIPE, check=False)
```

左边的红色代表这一行没有被覆盖。

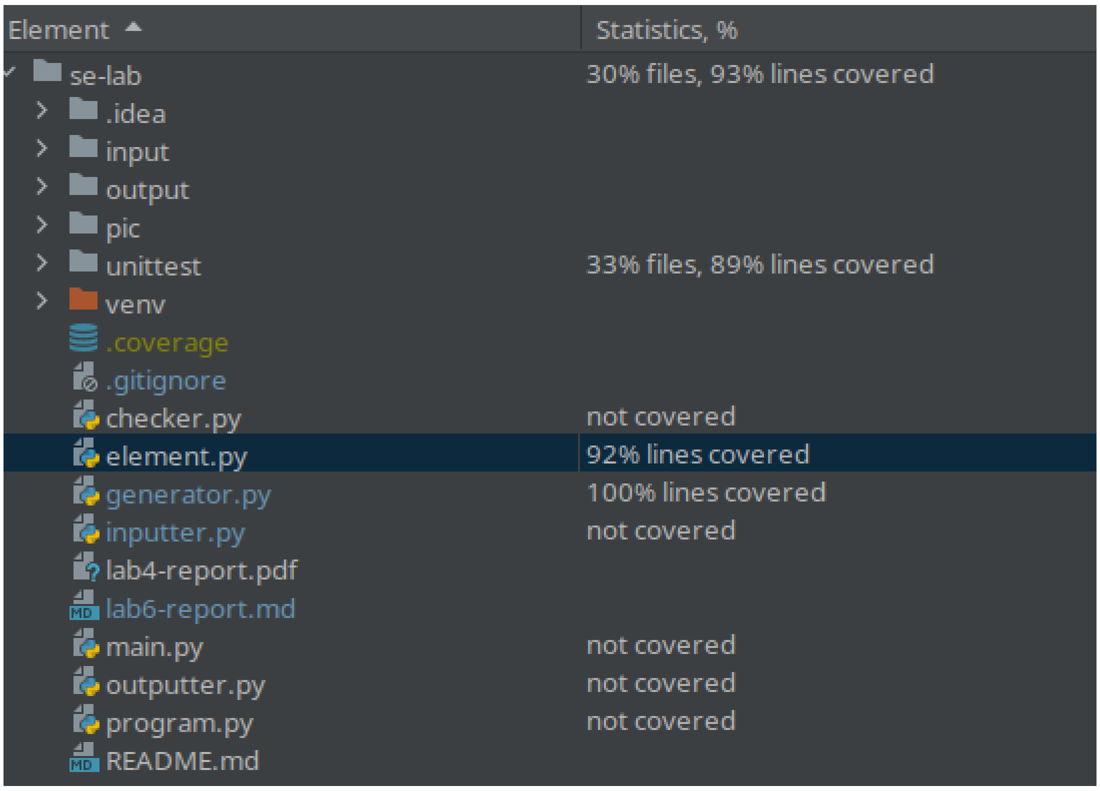
3. 接下来我们对 `generator.py` 进行测试：
   测试目的是为了测试generator生成器的实现正确性。
   测试用例，预期输出见下列代码

```python
class MyTestCase(unittest.TestCase):
    def test_int(self):
        self.assertTrue(1 <= int(Generator.random_int(1, 4)) <= 4)
        self.assertTrue(11 <= int(Generator.random_int(11, 45)) <= 45)
        self.assertTrue(114 <= int(Generator.random_int(114, 514)) <= 514)

    def test_char(self):
        char_1 = Generator.random_char()
        self.assertTrue(ord('a') <= ord(char_1) <= ord('z') or ord('A') <=
ord(char_1) <= ord('Z'))

    def test_str(self):
        str_1 = Generator.random_str(114, 514)
        self.assertIsInstance(str_1, str)
        for ch in str_1:
            self.assertTrue('a' <= ch <= 'z' or 'A' <= ch <= 'Z')

    def test_gen(self):
        test_format = [[('int', 1, 1), ('int', 4, 5), ('int', 1, 4)],
                       ['char', 'char', 'char'],
                       [('string', 19, 19), ('int', 8, 10)]]
        test_format = [[Element(elem) for elem in line] for line in
test_format]
        test_generator = Generator(test_format)
        test_str = test_generator.gen_test()
        for lineno, line in enumerate(test_str.strip(' \n').split('\n')):
            for elemno, elem in enumerate(line.strip(' ').split(' ')):
                if lineno == 0:
                    if elemno == 0:
                        self.assertEqual(int(elem), 1)
                    elif elemno == 1:
                        self.assertTrue(4 <= int(elem) <= 5)
                    elif elemno == 2:
                        self.assertTrue(1 <= int(elem) <= 4)
                    else:
                        self.assertTrue(False)
                elif lineno == 1:
                    if 0 <= elemno <= 2:
                        self.assertTrue(ord('a') <= ord(elem) <= ord('z') or ord('A') <= ord(elem) <= ord('Z'))
                    else:
                        self.assertTrue(False)
                elif lineno == 2:
                    if elemno == 0:
                        self.assertIsInstance(elem, str)
                        self.assertTrue(len(elem) == 19)
                        for ch in elem:
                            self.assertTrue('a' <= ch <= 'z' or 'A' <= ch <= 'Z')
                    elif elemno == 1:
                        self.assertTrue(8 <= int(elem) <= 10)
                    else:
                        self.assertTrue(False)
                else:
                    self.assertTrue(False)


if __name__ == '__main__':
    unittest.main()
```

在测试的时候，我发现此前的 `random_int()` 函数的实现是错误的，在 `test_int()` 中的第一条语句报错，输出了 `5`。我发现源码中我实现如下：`return random.randint(lower, upper + 1)` ，去查阅手册发现 `random.randint(a, b)` 是在 [a, b] 区间中随机取值，因此之前的实现有误。

修改后，通过了所有单元测试。

测试覆盖率：`generator.py` 100%, `element.py` 92%



```
Element                 Statistics, %
se-lab                  30% files, 93% lines covered
  .idea
  input
  output
  pic
  unittest              33% files, 89% lines covered
  venv
  .coverage
  .gitignore
  checker.py            not covered
  element.py            92% lines covered
  generator.py          100% lines covered
  inputter.py           not covered
  lab4-report.pdf
  lab6-report.md
  main.py               not covered
  outputter.py          not covered
  program.py            not covered
  README.md
```

4. 接下来对 `checker.py` 进行测试
   测试目的：测试checker.py的实现正确。
   由于checker的特殊性：带有随机性，不能保证完全正确，我选取了输出固定的unittest进行测试：

```python
class MyTestCase(unittest.TestCase):
    def test_checker(self):
        generator_1 = Generator([])
        progs_1 = [Program('tests/inputter1/48762087.cpp'),
                   Program('tests/inputter1/127473352.cpp'),
                   Program('tests/inputter1/134841308.cpp')]
        eq_1, neq_1 = Checker.check_list(progs_1, generator_1)
        for pair in eq_1:
            self.assertTrue(set(pair) == set(progs_1[1:3]))
        for pair in neq_1:
            self.assertTrue(set(pair) == set(progs_1[0:2]) or
                            set(pair) == set(progs_1[0:3:2]))
```

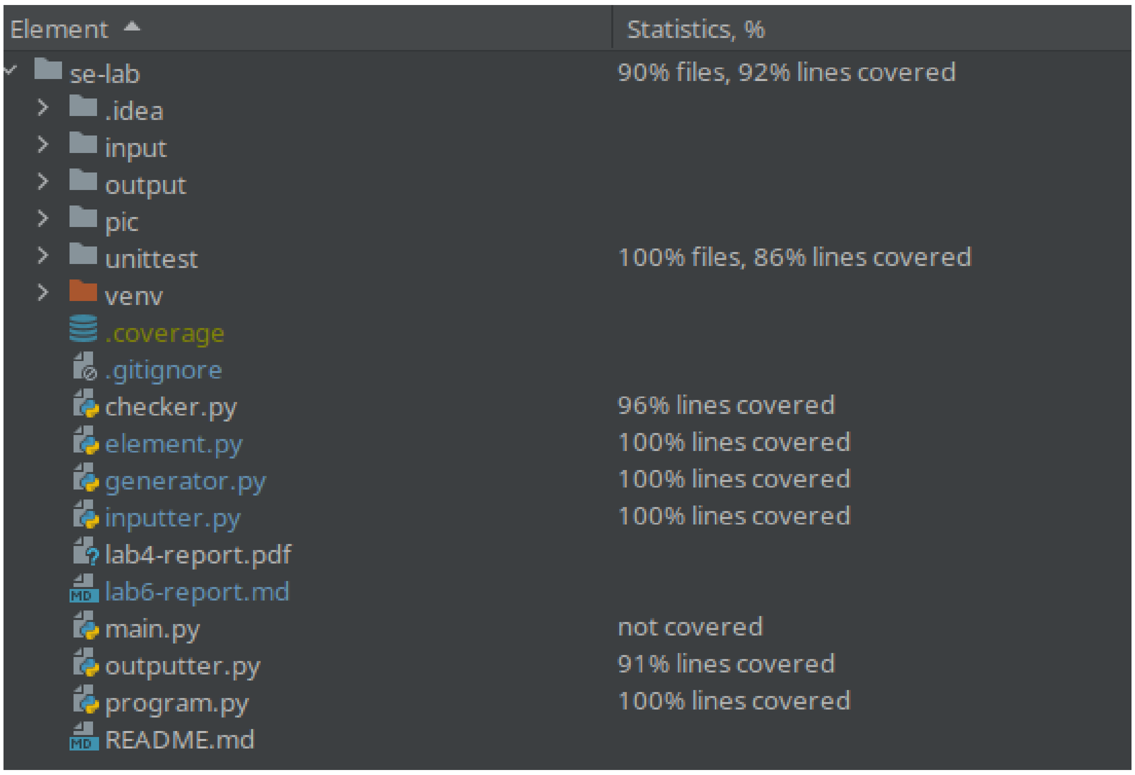
   测试结果是通过了所有测试样例，对checker.py有96%的覆盖率

5. 最后对 `outputter.py` 进行测试
   测试目的：测试output模块是否有正确输出。

```python
class MyTestCase(unittest.TestCase):
    def test_output(self):
        program_1 = Program('tests/inputter1/134841308.cpp')
        program_2 = Program('tests/inputter1/48762087.cpp')
        program_3 = Program('tests/inputter1/84822638.cpp')
        eq_list1 = [[program_1, program_2]]
        neq_list1 = [[program_1, program_3], [program_2, program_3]]
        out_1 = Outputter(eq_list1, neq_list1, 'tests')
        if os.path.exists('tests/equal.csv'):
            os.remove('tests/equal.csv')
        if os.path.exists('tests/inequal.csv'):
            os.remove('tests/inequal.csv')
        out_1.write_csv()
        self.assertTrue(os.path.exists('tests/equal.csv'))
        self.assertTrue(os.path.exists('tests/inequal.csv'))
        if os.path.exists('tests/equal.csv'):
            os.remove('tests/equal.csv')
        if os.path.exists('tests/inequal.csv'):
            os.remove('tests/inequal.csv')
```

   成功通过了所有测试，Output模块覆盖率为91%.

6. 综上，我们的单元测试对每个模块完成，且文件覆盖率达到90%，语句覆盖率达到92%：



Element	Statistics, %
se-lab	90% files, 92% lines covered
.idea	
input	
output	
pic	
unittest	100% files, 86% lines covered
venv	
.coverage	
.gitignore	
checker.py	96% lines covered
element.py	100% lines covered
generator.py	100% lines covered
inputter.py	100% lines covered
lab4-report.pdf	
lab6-report.md	
main.py	not covered
outputter.py	91% lines covered
program.py	100% lines covered
README.md	

三、集成测试

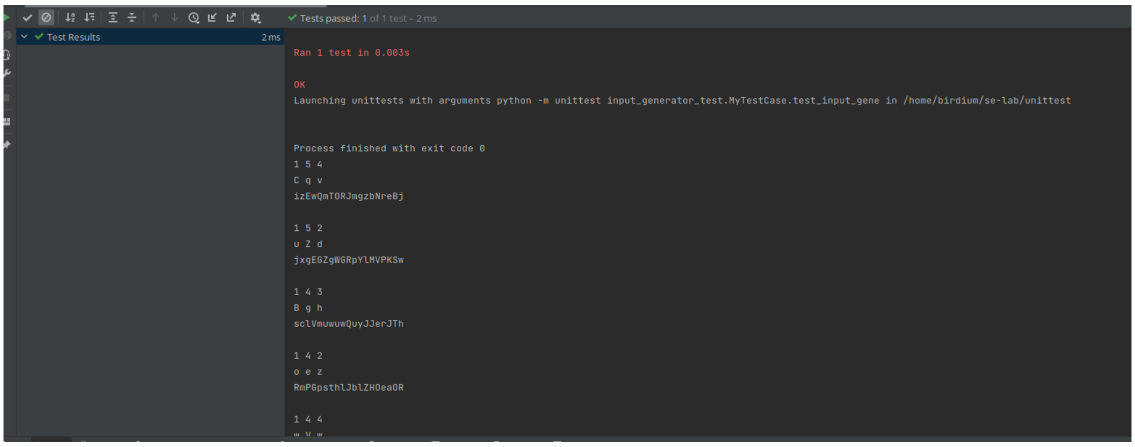
1. 测试目的：测试模块之间交互的正确性
   测试对象：等价判断工具
   测试环境、工具：同上
   测试方法：自底向上的测试方法
2. 首先将inputter和generator两个模块集成起来测试：
   测试目的：测试inputter模块和generator模块之间的交互

```
    def test_input_gene(self):
        inputter_1 = Inputter(r"tests/inputter4")
        format_1 = inputter_1.get_format()
        generator_1 = Generator(format_1)
        for _ in range(10):
            test_str = generator_1.gen_test()
            print(test_str)
            for lineno, line in enumerate(test_str.strip(' \n').split('\n')):
                for elemno, elem in enumerate(line.strip(' ').split(' ')):
                    if lineno == 0:
                        if elemno == 0:
                            self.assertEqual(int(elem), 1)
                        elif elemno == 1:
                            self.assertTrue(4 <= int(elem) <= 5)
                        elif elemno == 2:
                            self.assertTrue(1 <= int(elem) <= 4)
                        else:
                            self.assertTrue(False)
                    elif lineno == 1:
                        if 0 <= elemno <= 2:
                            self.assertTrue(ord('a') <= ord(elem) <= ord('z')
or ord('A') <= ord(elem) <= ord('Z'))
                        else:
```

```
              self.assertTrue(False)
          elif lineno == 2:
            if elemno == 0:
              self.assertIsInstance(elem, str)
              self.assertTrue(len(elem) == 19)
              for ch in elem:
                self.assertTrue('a' <= ch <= 'z' or 'A' <= ch <= 'Z')
            elif elemno == 1:
              self.assertTrue(8 <= int(elem) <= 10)
            else:
              self.assertTrue(False)
          else:
            self.assertTrue(False)
```

预期输出见代码，测试用例是一个只含有stdin_format.txt的文件夹，stdin_format.txt的内容同 `inputter.py` 单元测试中的内容

测试结果是通过了所有样例，实际输出如下：

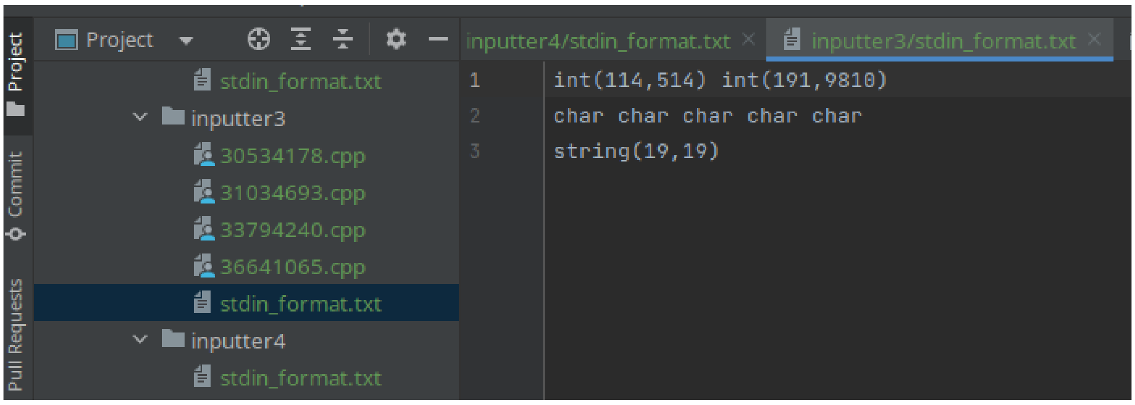


```
Ran 1 test in 0.003s

OK
Launching unittests with arguments python -m unittest input_generator_test.MyTestCase.test_input_gene in /home/birdium/se-lab/unittest


Process finished with exit code 0
1 5 4
C q v
izEwQmTORJmgzbNreBj

1 5 2
u Z d
jxgEGZgWGRpYlMVPKSw

1 4 3
B g h
sclVmuwuwQuyJJerJTh

1 4 2
o e z
RmPGpsthlJblZHOeaOR

1 4 4
```

3. 然后我们将checker集成进来测试，测试目的是检查从inputter到generator到checker这一过程中有没有出现问题

```
  def test_input_checker(self):
    inputter_1 = Inputter(r"tests/inputter3")
    format_1 = inputter_1.get_format()
    programs_1 = inputter_1.get_programs()
    generator_1 = Generator(format_1)
    eq_1, neq_1 = Checker.check_list(programs_1, generator_1)
    self.assertEqual(len(eq_1) + len(neq_1), len(programs_1) * (len(programs_1) - 1) / 2)
```

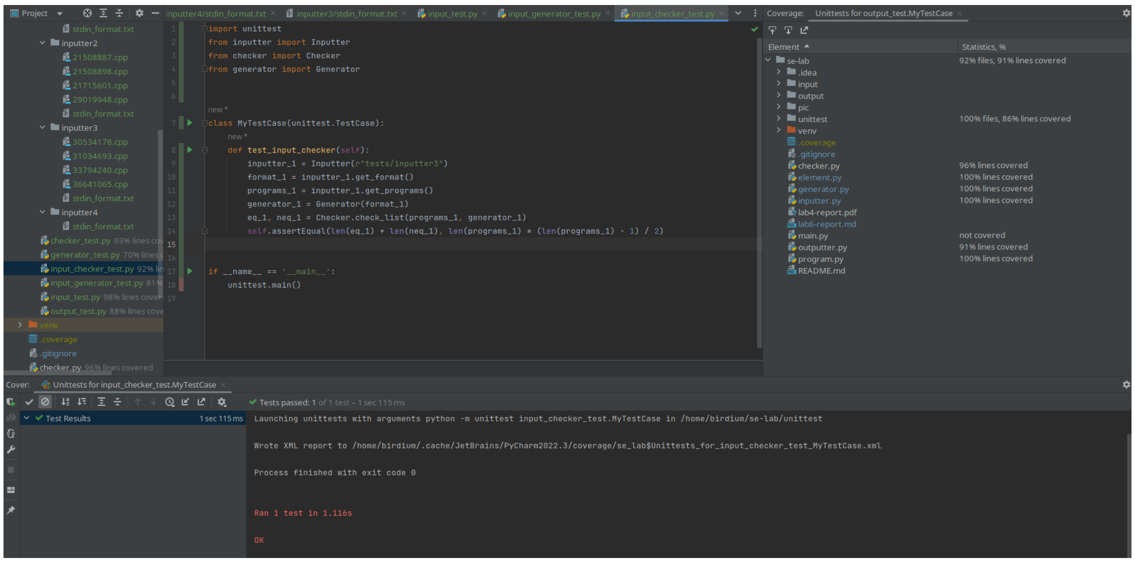
代码中使用了一个文件夹作为测试用例，文件夹结构如下：



```
int(114,514) int(191,9810)
char char char char char
string(19,19)
```

预期输出是输出的等价程序对和不等价程序对加起来和为 C_n^2

实际输出是程序给出了正确的结果。



```python
import unittest
from inputter import Inputter
from checker import Checker
from generator import Generator


class MyTestCase(unittest.TestCase):
    def test_input_checker(self):
        inputter_1 = Inputter(r"tests/inputter3")
        format_1 = inputter_1.get_format()
        programs_1 = inputter_1.get_programs()
        generator_1 = Generator(format_1)
        eq_1, neq_1 = Checker.check_list(programs_1, generator_1)
        self.assertEqual(len(eq_1) + len(neq_1), len(programs_1) * (len(programs_1) - 1) / 2)


if __name__ == '__main__':
    unittest.main()
```

Coverage: Unittests for output_test.MyTestCase

Element	Statistics, %
se-lab	92% files, 91% lines covered
.idea	
input	
output	
pic	
unittest	100% files, 86% lines covered
venv	
.coverage	
.gitignore	
checker.py	96% lines covered
element.py	100% lines covered
generator.py	100% lines covered
inputter.py	100% lines covered
lab4-report.pdf	
lab6-report.md	
main.py	not covered
outputter.py	91% lines covered
program.py	100% lines covered
README.md	

Tests passed: 1 of 1 test – 1 sec 115 ms

```
Launching unittests with arguments python -m unittest input_checker_test.MyTestCase in /home/birdium/se-lab/unittest

Wrote XML report to /home/birdium/.cache/JetBrains/PyCharm2022.3/coverage/se_lab$Unittests_for_input_checker_test_MyTestCase.xml

Process finished with exit code 0


Ran 1 test in 1.116s

OK
```

综上，我们完成了两个集成测试。