CS434 Software Engineering and Testing

Tutorial #1 (08/01/2024)

Q1. Illustrate an execution of the following C code where we obtain statement (line) coverage as: (i) 100%, and (ii) < 100%. Using software tools (**`gcc`**, **`gcov,lcov, genhtml`**) generate a report. Why is code coverage so important?

```
//filename: fact.c
#include <stdio.h>
int factorial(int n)
{
    if (n == 0)
        return 1;
    else
        return n * factorial(n - 1);
}
int main()
{
    int n;
    printf("Enter a non-negative integer: ");
    scanf("%d", &n);
    printf("The factorial of %d is %d.\n", n, factorial(n));
    return 0;
}
```

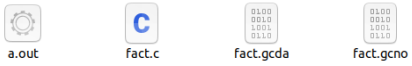
Answer:

- Step 1 [Instrumentation]:
  `gcc -Wall -fprofile-arcs -ftest-coverage fact.c`

- Step 2 [Execution #1 of the instrumented a.out]:
  `./a.out`

  ```
  Enter a non-negative integer: 5
  The factorial of 5 is 120.
  ```

- Step 3 [Show statement (line) coverage coverage]
  `gcov fact.c`

```
File 'fact.c'
Lines executed:100.00% of 9
Creating 'fact.c.gcov'
```

- Step 4 [View `fact.c.gcov`]

```
  -:    0:Source:fact.c
  -:    0:Graph:fact.gcno
  -:    0:Data:fact.gcda
  -:    0:Runs:1
  -:    1:#include <stdio.h>
  6:    2:int factorial(int n)
  -:    3:{
  6:    4:    if (n == 0)
  1:    5:        return 1;
  -:    6:    else
  5:    7:        return n * factorial(n - 1);
  -:    8:}
  1:    9:int main()
  -:   10:{
  -:   11:    int n;
  1:   12:    printf("Enter a non-negative integer: ");
  1:   13:    scanf("%d", &n);
  1:   14:    printf("The factorial of %d is %d.\n", n, factorial(n));
  1:   15:    return 0;
  -:   16:}
```

- Step 5 [Cleanup results of previous execution]
  `rm a.out *.gcno *.gcov *.gcda`

- Step 6 [Re-create instrumented `a.out`]
  `gcc -Wall -fprofile-arcs -ftest-coverage fact.c`

- Step 7 [Execution #2 of the instrumented a.out]:
  `./a.out`

```
Enter a non-negative integer: 0
The factorial of 0 is 1.
```

- Step 8 [Show statement (line) coverage coverage]
  `gcov fact.c`

```
File 'fact.c'
Lines executed:88.89% of 9
Creating 'fact.c.gcov'
```

- Step 9 [View `fact.c.gcov`]

```
    -:    0:Source:fact.c
    -:    0:Graph:fact.gcno
    -:    0:Data:fact.gcda
    -:    0:Runs:1
    -:    1:#include <stdio.h>
    1:    2:int factorial(int n)
    -:    3:{
    1:    4:    if (n == 0)
    1:    5:        return 1;
    -:    6:    else
#####:    7:        return n * factorial(n - 1);
    -:    8:}
    1:    9:int main()
    -:   10:{
    -:   11:    int n;
    1:   12:    printf("Enter a non-negative integer: ");
    1:   13:    scanf("%d", &n);
    1:   14:    printf("The factorial of %d is %d.\n", n, factorial(n));
    1:   15:    return 0;
    -:   16:}
```

- Step 10 [Use `lcov + genhtml`] (for visualization)
  `lcov --capture --directory . --output-file coverage.info`
  `genhtml coverage.info`

```
Reading data file coverage.info
…
Overall coverage rate:
  lines......: 88.9% (8 of 9 lines)
  functions..: 100.0% (2 of 2 functions)
```

LCOV - code coverage report

Current view:	top level		Hit	Total	Coverage
Test:	coverage.info	Lines:	8	9	88.9 %
Date:	2024-01-09 00:10:32	Functions:	2	2	100.0 %

Directory	Line Coverage		Functions	
tut		88.9 %	8 / 9	100.0 %	2 / 2

Generated by: LCOV version 1.14

LCOV - code coverage report

Current view:	top level - tut - fact.c (source / functions)		Hit	Total	Coverage
Test:	coverage.info	Lines:	8	9	88.9 %
Date:	2024-01-09 00:10:32	Functions:	2	2	100.0 %

```
      Line data    Source code
    1            : #include <stdio.h>
    2          1 : int factorial(int n)
    3            : {
    4          1 :     if (n == 0)
    5          1 :         return 1;
    6            :     else
    7          0 :         return n * factorial(n - 1);
    8            : }
    9          1 : int main()
   10            : {
   11            :     int n;
   12          1 :     printf("Enter a non-negative integer: ");
   13          1 :     scanf("%d", &n);
   14          1 :     printf("The factorial of %d is %d.\n", n, factorial(n));
   15          1 :     return 0;
   16            : }
```

Generated by: LCOV version 1.14

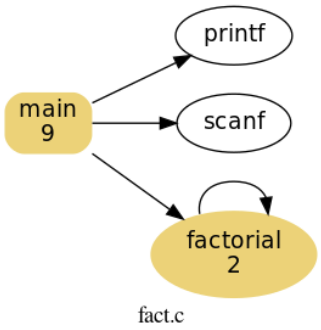
Q2. Show the function call graph of `fact.c` in textual mode and graphical mode.

Answer:
Textual mode: `cflow fact.c`

```
main() <int main () at fact.c:9>:
    printf()
    scanf()
    factorial() <int factorial (int n) at fact.c:2> (R):
        factorial() <int factorial (int n) at fact.c:2> (recursive: see 4)
```

Graphical code: `cflow2dot -i fact.c`

```
cflow2dot
found cflow at: /usr/bin/cflow
found dot at: /usr/bin/dot
This may take some time... ...
svg produced successfully from dot.
```

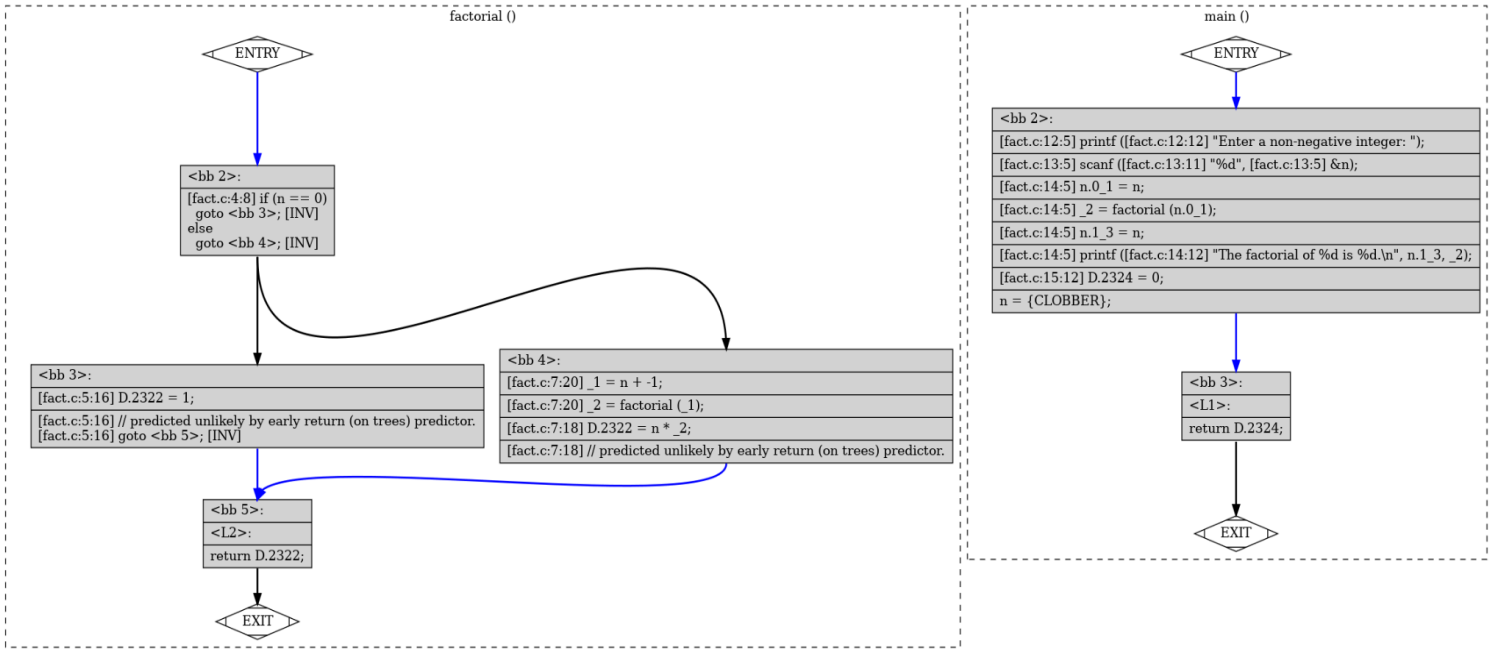


Q3. Show the control flow graph of the source file `fact.c`

Answer:

- Step 1 [Generate control flow graph]
  `gcc -fdump-tree-cfg-graph-lineno fact.c`

- Step 2 [Convert .dot to png]



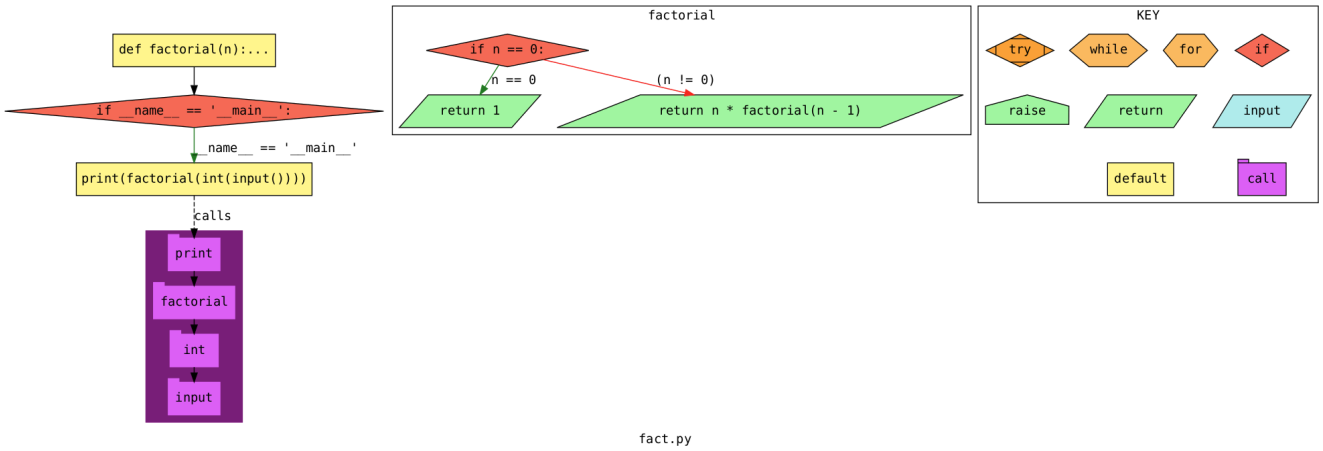
`dot -Tpng fact.c.012t.cfg.dot -o fact.c.png`

Q4. Generate (i) the control flow graph for the following Python code (`fact.py`)

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
if __name__ == "__main__":
      print(factorial(int(input())))
```

Answer:

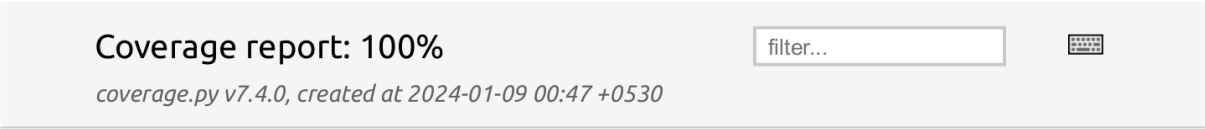
`py2cfg fact.py`



Q5. For `fact.py` Illustrate an execution where we obtain statement (line) coverage as: (i) 100%, and (ii) < 100%. Using software tools (**`coverage`**) generate a report. Why is code coverage so important?

Answer:

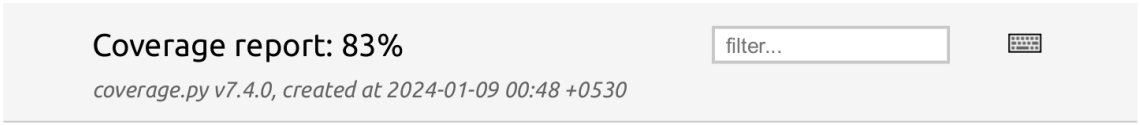
```
coverage run fact.py
5
120
coverage html
```



Module	statements	missing	excluded	coverage
fact.py	6	0	0	100%
Total	**6**	**0**	**0**	**100%**

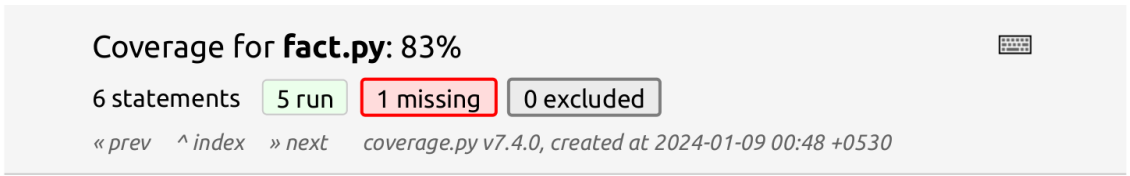
coverage.py v7.4.0, created at 2024-01-09 00:47 +0530

```
coverage run fact.py
0
1
coverage html
```



Module	statements ↑	missing	excluded	coverage
fact.py	6	1	0	83%
Total	**6**	**1**	**0**	**83%**

coverage.py v7.4.0, created at 2024-01-09 00:48 +0530




```
1 def factorial(n):
2     if n == 0:
3         return 1
4     else:
5         return n * factorial(n-1)
6
7 if __name__ == "__main__":
8         print(factorial(int(input())))
```

« prev ^ index » next coverage.py v7.4.0, created at 2024-01-09 00:48 +0530

Q6. Automated unit test-case generation for `fact.py` (main removed)

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

Answer: fact.py

```
pynguin --project-path . --output-path . --module-name fact -v
```

```
[01:03:30] INFO     Start Pynguin Test Generation…                                  generator.py:10
           INFO     Collecting static constants from module under test              generator.py:20
           INFO     No constants found                                              generator.py:21
           INFO     Setting up runtime collection of constants                      generator.py:22
           INFO     Analyzed project to create test cluster                           module.py:131
           INFO     Modules:       1                                                  module.py:131
           INFO     Functions:     1                                                  module.py:132
           INFO     Classes:      11                                                  module.py:132
           INFO     Using seed 1704742409855393620                                  generator.py:19
           INFO     Using strategy: Algorithm.DYNAMOSA                generationalgorithmfactory.py:30
           INFO     Instantiated 3 fitness functions                  generationalgorithmfactory.py:39
           INFO     Using CoverageArchive                             generationalgorithmfactory.py:34
           INFO     Using selection function: Selection.TOURNAMENT_SELECTION  generationalgorithmfactory.py:32
           INFO     No stopping condition configured!                 generationalgorithmfactory.py:11
           INFO     Using fallback timeout of 600 seconds             generationalgorithmfactory.py:12
           INFO     Using crossover function: SinglePointRelativeCrossOver  generationalgorithmfactory.py:33
           INFO     Using ranking function: RankBasedPreferenceSorting  generationalgorithmfactory.py:35
           INFO     Start generating test cases                                     generator.py:51
           INFO     Initial Population, Coverage: 1.000000                      searchobserver.py:7
           INFO     Algorithm stopped before using all resources.                   generator.py:52
           INFO     Stop generating test cases                                      generator.py:52
           INFO     Start generating assertions                                     generator.py:59
           INFO     Setup mutation controller                                 mutationadapter.py:7
           INFO     Build AST for fact                                        mutationadapter.py:6
           INFO     Mutate module fact                                        mutationadapter.py:6
           INFO     Generated 9 mutants                                       mutationadapter.py:7
           INFO     Running tests on mutant   1/9                         assertiongenerator.py:29
           INFO     Running tests on mutant   2/9                         assertiongenerator.py:29
           INFO     Running tests on mutant   3/9                         assertiongenerator.py:29
           INFO     Running tests on mutant   4/9                         assertiongenerator.py:29
           INFO     Running tests on mutant   5/9                         assertiongenerator.py:29
           INFO     Running tests on mutant   6/9                         assertiongenerator.py:29
           INFO     Running tests on mutant   7/9                         assertiongenerator.py:29
           INFO     Running tests on mutant   8/9                         assertiongenerator.py:29
           INFO     Running tests on mutant   9/9                         assertiongenerator.py:29
           INFO     Mutant 0 killed by Test(s): 1                         assertiongenerator.py:37
           INFO     Mutant 1 killed by Test(s): 1                         assertiongenerator.py:37
           INFO     Mutant 2 killed by Test(s): 1                         assertiongenerator.py:37
           INFO     Mutant 3 killed by Test(s): 0, 1                      assertiongenerator.py:37
           INFO     Mutant 4 killed by Test(s): 1                         assertiongenerator.py:37
           INFO     Mutant 5 killed by Test(s): 1                         assertiongenerator.py:37
           INFO     Mutant 6 killed by Test(s): 0, 1                      assertiongenerator.py:37
           INFO     Mutant 7 killed by Test(s): 0, 1                      assertiongenerator.py:37
           INFO     Mutant 8 killed by Test(s): 1                         assertiongenerator.py:37
           INFO     Number of Surviving Mutant(s): 0 (Mutants: )          assertiongenerator.py:38
           INFO     Calculating resulting FinalBranchCoverage                       generator.py:43
           INFO     Written 2 test cases to /home/shouvick/cache_test/tut/test_fact.py  generator.py:70
           INFO     Writing statistics                                           statistics.py:36
           INFO     Stop Pynguin Test Generation…                                   generator.py:11
```

```python
# Test cases automatically generated by Pynguin (https://www.pynguin.eu).
# Please check them before you use them.
import pytest
import fact as module_0


def test_case_0():
    bool_0 = False
    var_0 = module_0.factorial(bool_0)
    assert var_0 == 1


@pytest.mark.xfail(strict=True)
def test_case_1():
    bool_0 = False
    set_0 = {bool_0}
    module_0.factorial(set_0)
```

Q7. Explore the following tools:

- **`gcov`** (https://en.wikipedia.org/wiki/Gcov)
- **`py2cfg`** (https://gitlab.com/classroomcode/py2cfg)
- **`cflow2dot`** (https://pypi.org/project/pycflow2dot)
- **`cflow`** (https://www.gnu.org/software/cflow)
- **`coverage`** (https://coverage.readthedocs.io/en/latest)
- **`pynguin`** (https://www.pynguin.eu)

Q8. Explore more:

- Python tools (https://awesome-python.com)
- Generate the control flow graph (CFG) using **`gcc`** from C source code.
    - https://stackoverflow.com/questions/16393985/getting-control-flow-graph-from-ansi-c-code/17844310#17844310
    - https://swtv.kaist.ac.kr/courses/cs492-fall18/part1-coverage/lec5.5-cfg-generation.pdf