Homework #1: Manual Test-Case Generation

Due date: 2023-05-01 (23:59:00)

In the field of software engineering, **generating test-cases for testing the target program is a very common but important task**. Hence, through this assignment, we want each student to experience this test-case generation.

The goal of this homework is to **manually generate the test-cases that maximize code coverage (e.g., the number of covered branches**) which is one of the well-known metrics for evaluating the quality of the test-cases. Using an example in Figure 1, let's calculate the number of covered branches. First, in general, the number of branches in the program is twice the number of if (or while) statements. As there are two if statements, the total number of branches are 4. To cover all the branches in the toy program of Figure 1, we need only two-test-cases to be executed. For example, when the first test-case is that x is 90 and y =25, then we can cover two true branches; when the second test-case is that x is 0 and y is 0, we can cover two false branches, thereby successfully covering all the four branches. In our homework, we try to generate the test-cases that increase the number of covered branches for a real-world open-source C program.

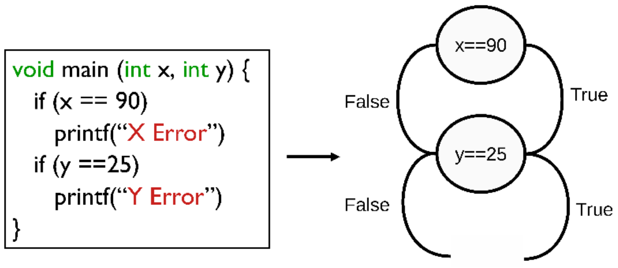

```
void main (int x, int y) {
  if (x == 90)
    printf("X Error")
  if (y ==25)
    printf("Y Error")
}
```

Figure 1 Examples for the number of covered branches

The target program that you will test is **'gawk-5.2.1'**, which is a program for selecting particular records in a file and performing operations upon them. For the program, the more the number of branches covered by executing the program with your manual test-cases, the higher your homework score will be. We will automatically calculate the number of covered branches using gcov, a famous coverage calculation tool. Also, I attach the manual of the "gawk" program for effectively increasing the number of covered branches for the 'gawk' program.

The maximum number of test cases that you can make is 30 (1<= # of test cases<=30). The number of covered branches varies depending on the test-cases that you made, so we will give you the score according to that value (the number of covered branches). All you need to do is to generate up to 30 test-cases used in the "gawk" command and write in studentid_testcases.txt. We are going to explain how to download and run homework files, so please follow those instructions and if you have any questions, please contact TA by email.

Version & Environment

- Benchmark: gawk-5.2.1
- Evaluation Environment: **Linux Ubuntu-18.04.5 (swji of In-Ui-Ye-Ji server)**

** For any account-related question, please send an email to inuiyeji-skku@googlegroups.com including your student id and public IP.

Grading

- The testcases will be evaluated three times and the average will be your final coverage rate.
- Each coverage rate will be sorted in ascending order, and the scores will be given in groups of 10 people. The first group gets 10, and the lower groups get 1 point deducted subsequently.

** Each testcases needs to be done in 3 seconds. Otherwise, the time-outed command will be ignored.

*** Covered branches may vary depending on Ubuntu version. Test your testcases on swji server.

Submission

- just submit **studentid_testcases.txt** and upload it to i-campus (change **studentid** to your id).
- Due date: 2023-05-01, 23:59:00 (No late submission.)

** Caution: If Windows newline “\r” is in testcases, they return errors. Do not write your testcases file on Windows OS.

Introductions

1. Build benchmark by gcov

First, download the *.zip* file from i-campus and extract the *.zip* file.

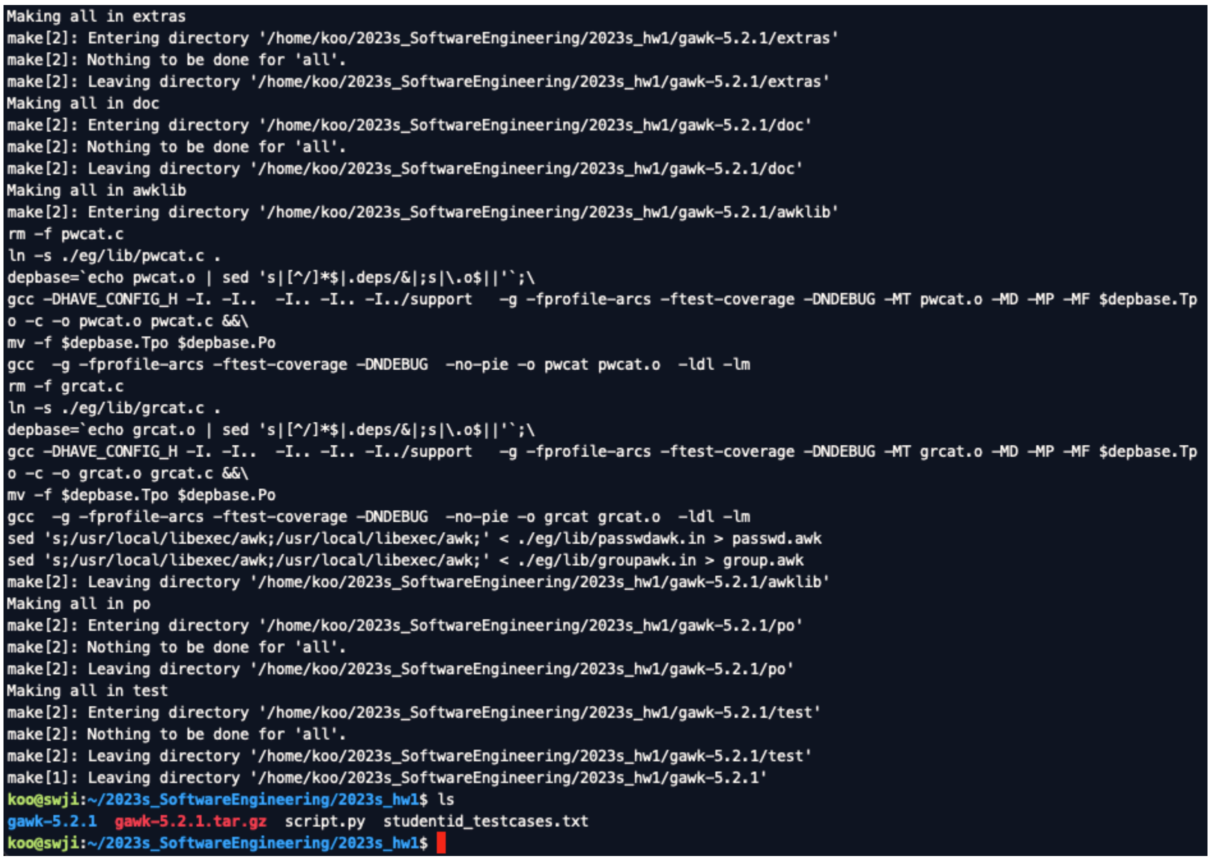
```
$ unzip 2023s_hw1.zip
```

After the given *.zip* file extraction,

```
$ cd 2023s_hw1
$ python3 script.py --build true
```

** "--build" option builds gawk and gcov at the same time.

If you build it well, terminal will show the screen below.



```
Making all in extras
make[2]: Entering directory '/home/koo/2023s_SoftwareEngineering/2023s_hw1/gawk-5.2.1/extras'
make[2]: Nothing to be done for 'all'.
make[2]: Leaving directory '/home/koo/2023s_SoftwareEngineering/2023s_hw1/gawk-5.2.1/extras'
Making all in doc
make[2]: Entering directory '/home/koo/2023s_SoftwareEngineering/2023s_hw1/gawk-5.2.1/doc'
make[2]: Nothing to be done for 'all'.
make[2]: Leaving directory '/home/koo/2023s_SoftwareEngineering/2023s_hw1/gawk-5.2.1/doc'
Making all in awklib
make[2]: Entering directory '/home/koo/2023s_SoftwareEngineering/2023s_hw1/gawk-5.2.1/awklib'
rm -f pwcat.c
ln -s ./eg/lib/pwcat.c .
depbase=`echo pwcat.o | sed 's|[^/]*$|.deps/&|;s|\.o$||'`;\
gcc -DHAVE_CONFIG_H -I. -I..  -I.. -I.. -I../support   -g -fprofile-arcs -ftest-coverage -DNDEBUG -MT pwcat.o -MD -MP -MF $depbase.Tp
o -c -o pwcat.o pwcat.c &&\
mv -f $depbase.Tpo $depbase.Po
gcc  -g -fprofile-arcs -ftest-coverage -DNDEBUG  -no-pie -o pwcat pwcat.o  -ldl -lm
rm -f grcat.c
ln -s ./eg/lib/grcat.c .
depbase=`echo grcat.o | sed 's|[^/]*$|.deps/&|;s|\.o$||'`;\
gcc -DHAVE_CONFIG_H -I. -I..  -I.. -I.. -I../support   -g -fprofile-arcs -ftest-coverage -DNDEBUG -MT grcat.o -MD -MP -MF $depbase.Tp
o -c -o grcat.o grcat.c &&\
mv -f $depbase.Tpo $depbase.Po
gcc  -g -fprofile-arcs -ftest-coverage -DNDEBUG  -no-pie -o grcat grcat.o  -ldl -lm
sed 's;/usr/local/libexec/awk;/usr/local/libexec/awk;' < ./eg/lib/passwdawk.in > passwd.awk
sed 's;/usr/local/libexec/awk;/usr/local/libexec/awk;' < ./eg/lib/groupawk.in > group.awk
make[2]: Leaving directory '/home/koo/2023s_SoftwareEngineering/2023s_hw1/gawk-5.2.1/awklib'
Making all in po
make[2]: Entering directory '/home/koo/2023s_SoftwareEngineering/2023s_hw1/gawk-5.2.1/po'
make[2]: Nothing to be done for 'all'.
make[2]: Leaving directory '/home/koo/2023s_SoftwareEngineering/2023s_hw1/gawk-5.2.1/po'
Making all in test
make[2]: Entering directory '/home/koo/2023s_SoftwareEngineering/2023s_hw1/gawk-5.2.1/test'
make[2]: Nothing to be done for 'all'.
make[2]: Leaving directory '/home/koo/2023s_SoftwareEngineering/2023s_hw1/gawk-5.2.1/test'
make[1]: Leaving directory '/home/koo/2023s_SoftwareEngineering/2023s_hw1/gawk-5.2.1'
koo@swji:~/2023s_SoftwareEngineering/2023s_hw1$ ls
gawk-5.2.1  gawk-5.2.1.tar.gz  script.py  studentid_testcases.txt
koo@swji:~/2023s_SoftwareEngineering/2023s_hw1$
```

2. Make testcases

Open studentid_testcases.txt and write testcases one per line.

(In that text file, there will be two examples.)

3. Check the total coverage of your all testcases

```
$ cd ~/2023s_hw1/
$ python script.py studentid_testcases.txt
```

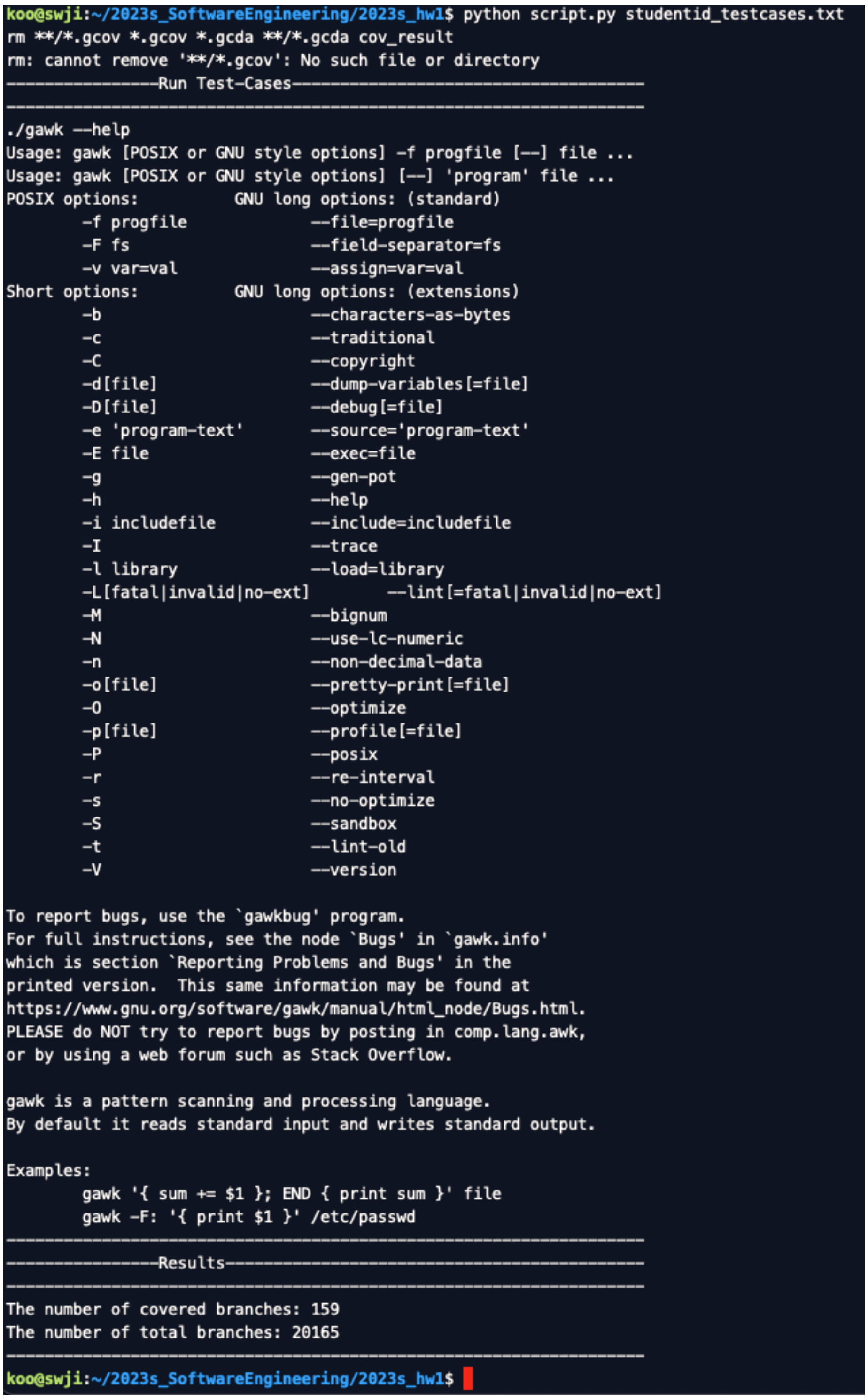
** The way script.py works **

1. If you give ***studentid_testcases.txt*** as input, ***script.py*** will run all the commands in the ***studentid_testcases.txt*** file.

(testcase: command that executes the *gawk* binary file)

e.g. ./gawk --help

2. After a binary file is executed (by each testcase), *.gcda* files are automatically created for source codes in ***2023s_hw1/gawk-5.2.1***
(cannot read *.gcda* file with “cat” or “vi” command)
3. Then it executes gcov and gcov creates *.gcov* file by reading created *.gcda* files.
4. Finally, it reads the created *.gcov* files and calculates the number of covered branches.



```
koo@swji:~/2023s_SoftwareEngineering/2023s_hw1$ python script.py studentid_testcases.txt
rm **/*.gcov *.gcov *.gcda **/*.gcda cov_result
rm: cannot remove '**/*.gcov': No such file or directory
---------------Run Test-Cases-------------------------------------
------------------------------------------------------------------
./gawk --help
Usage: gawk [POSIX or GNU style options] -f progfile [--] file ...
Usage: gawk [POSIX or GNU style options] [--] 'program' file ...
POSIX options:          GNU long options: (standard)
        -f progfile             --file=progfile
        -F fs                   --field-separator=fs
        -v var=val              --assign=var=val
Short options:          GNU long options: (extensions)
        -b                      --characters-as-bytes
        -c                      --traditional
        -C                      --copyright
        -d[file]                --dump-variables[=file]
        -D[file]                --debug[=file]
        -e 'program-text'       --source='program-text'
        -E file                 --exec=file
        -g                      --gen-pot
        -h                      --help
        -i includefile          --include=includefile
        -I                      --trace
        -l library              --load=library
        -L[fatal|invalid|no-ext]        --lint[=fatal|invalid|no-ext]
        -M                      --bignum
        -N                      --use-lc-numeric
        -n                      --non-decimal-data
        -o[file]                --pretty-print[=file]
        -O                      --optimize
        -p[file]                --profile[=file]
        -P                      --posix
        -r                      --re-interval
        -s                      --no-optimize
        -S                      --sandbox
        -t                      --lint-old
        -V                      --version

To report bugs, use the `gawkbug' program.
For full instructions, see the node `Bugs' in `gawk.info'
which is section `Reporting Problems and Bugs' in the
printed version.  This same information may be found at
https://www.gnu.org/software/gawk/manual/html_node/Bugs.html.
PLEASE do NOT try to report bugs by posting in comp.lang.awk,
or by using a web forum such as Stack Overflow.

gawk is a pattern scanning and processing language.
By default it reads standard input and writes standard output.

Examples:
        gawk '{ sum += $1 }; END { print sum }' file
        gawk -F: '{ print $1 }' /etc/passwd
------------------------------------------------------------------
---------------Results--------------------------------------------
------------------------------------------------------------------
The number of covered branches: 159
The number of total branches: 20165
------------------------------------------------------------------
koo@swji:~/2023s_SoftwareEngineering/2023s_hw1$
```

** Your goal is to increase the number of covered branches!! **

After checking the results, you just submit a text file.

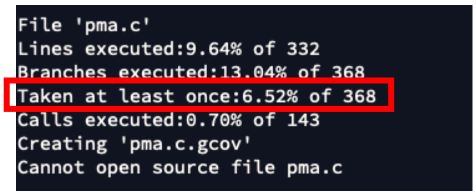
---

4. How to check the coverage of your each testcase

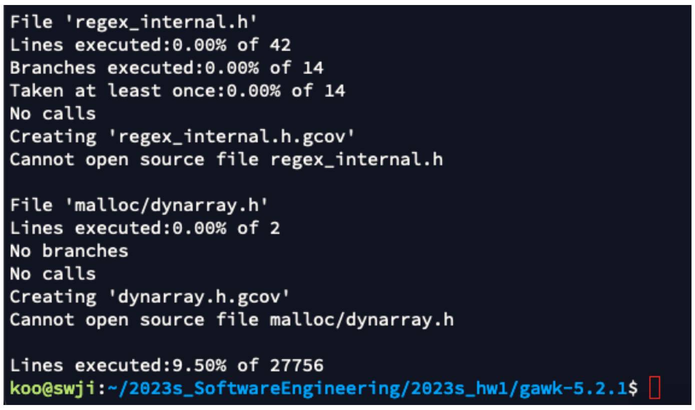
```
$ cd 2023s_hw1/gawk-5.2.1
$ ./gawk --help
$ gcov -b *.gcda **/*.gcda
```

Through this, you can check the number of covered branches for each testcase.

- In this case, ./gawk --help is the testcase (command).
- That command (testcase) creates the *.gcda* file for each source codes in 2023s_hw1/gawk-5.2.1 directory.
- gcov -b *.gcda **/*.gcda will check the number of covered branches by creating the *.gcov* files from *.gcda* files



```
File 'pma.c'
Lines executed:9.64% of 332
Branches executed:13.04% of 368
Taken at least once:6.52% of 368
Calls executed:0.70% of 143
Creating 'pma.c.gcov'
Cannot open source file pma.c
```



```
File 'regex_internal.h'
Lines executed:0.00% of 42
Branches executed:0.00% of 14
Taken at least once:0.00% of 14
No calls
Creating 'regex_internal.h.gcov'
Cannot open source file regex_internal.h

File 'malloc/dynarray.h'
Lines executed:0.00% of 2
No branches
No calls
Creating 'dynarray.h.gcov'
Cannot open source file malloc/dynarray.h

Lines executed:9.50% of 27756
koo@swji:~/2023s_SoftwareEngineering/2023s_hw1/gawk-5.2.1$
```

** The number of covered branches is calculated as follows **

For each file,

Taken at least once: 6.52% (← covered branch ratio) of 368 (← the number of total branches)

*The number of covered branches = 0.0652 * 368 = 24*

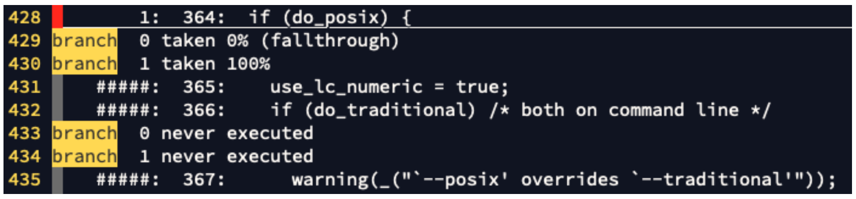
→ And we calculate the number of covered branches for all files and **add them all together**.

5. How to check the covered branches in each source code by your testcases

```
$ cd 2023s_hw1/gawk-5.2.1
$ ./gawk --help
$ gcov -b *.gcda **/*.gcda
$ vi main.c.gcov
```

- Command creates *.gcda* file

- gcov creates *.gcov* file, which you can read by “vi” or “cat” command
- *.gcov* file shows you which branch is covered
- You can search “branch” keyword in *.gcov* file



```
428          1:  364:  if (do_posix) {
429 branch  0 taken 0% (fallthrough)
430 branch  1 taken 100%
431      #####:  365:     use_lc_numeric = true;
432      #####:  366:     if (do_traditional) /* both on command line */
433 branch  0 never executed
434 branch  1 never executed
435      #####:  367:       warning(_("`--posix' overrides `--traditional'"));
```

In line 428: if (do_posix) → (*in line 428 in main.c.gcov*, if statement’s branch 0, branch 1)

- branch 0 taken 0% → means this branch is not covered
- branch 1 taken 100% → means this branch is covered

In line 432: if (do_traditional) → (*in line 432 in main.c.gcov*, if statement’s branch 0, branch 1)

- branch 0 never executed → means this branch is never executed
- branch 1 never executed → means this branch is never executed