Homework #1 Making Test Cases

Due date: 2022-05-08 (23:59:00)

Software developers use several coverage analysis programs to ensure that the developed software performs at a distributable level. The test cases used in these programs can verify whether the program works as expected, so we have to make good test cases to cover as many branches as possible.

In this assignment **we are going to make test cases to test whether the "grep" command covers as many branches as possible**. We can say that the more the number of covered branches, the better performance of the grep command.

We are going to check the coverage using **gcov**, the coverage test tool. Using the gcov tool, we can find out some basic performance statistics such as how often each line of code executes, what lines of code are actually executed, and how much computing time each section of code uses.

Your final goal is to increase the number of covered branches by writing test cases. There are several command options that use "grep". (if you enter *grep --help* in the Linux terminal, you can learn how to use "grep") **All you have to do is to create lists of test cases used in the "grep" command and write in *studentid_testcases.txt*.**

The maximum number of test cases is 30(1<= # of test cases<=30). The value of 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)***.

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: grep-3.7
- Coverage check tool: gcov
- Environment: Linux Ubuntu

Submission

- just submit **studentid_testcases.txt** and upload it to i-campus (change **studentid** to your id).
- Due date: 2022-05-08 , 23:59:00
- No late submission.

Instructions

1. Build benchmark by gcov

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

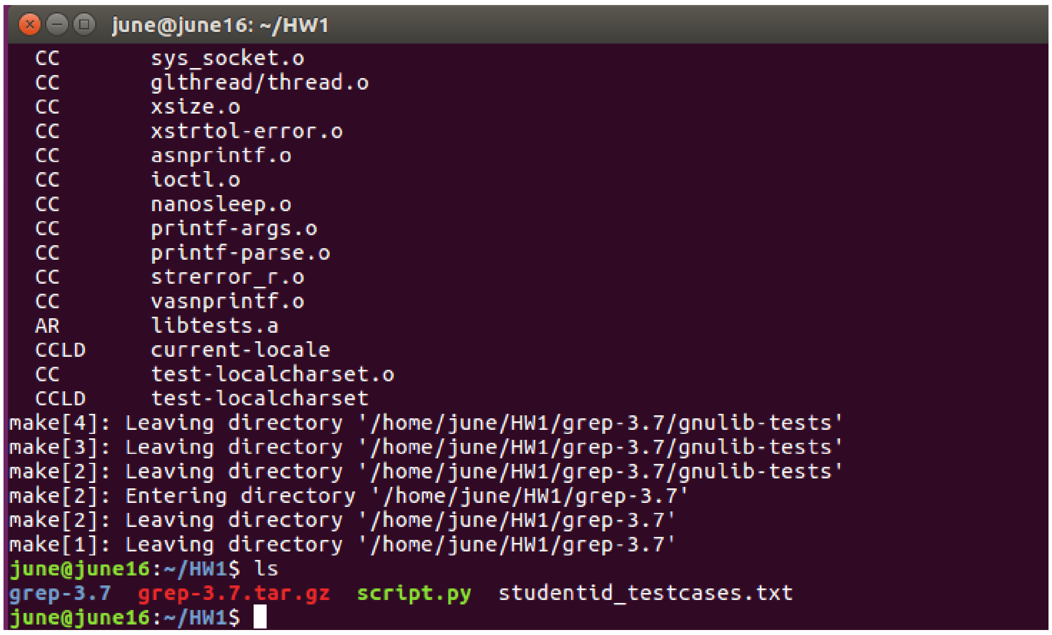
```
$ unzip HW1.zip
```

After the given *.zip* file extraction,

```
$ cd ~/HW1/
$ python3 script.py --build true
```

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

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



```
june@june16: ~/HW1
  CC       sys_socket.o
  CC       glthread/thread.o
  CC       xsize.o
  CC       xstrtol-error.o
  CC       asnprintf.o
  CC       ioctl.o
  CC       nanosleep.o
  CC       printf-args.o
  CC       printf-parse.o
  CC       strerror_r.o
  CC       vasnprintf.o
  AR       libtests.a
  CCLD     current-locale
  CC       test-localcharset.o
  CCLD     test-localcharset
make[4]: Leaving directory '/home/june/HW1/grep-3.7/gnulib-tests'
make[3]: Leaving directory '/home/june/HW1/grep-3.7/gnulib-tests'
make[2]: Leaving directory '/home/june/HW1/grep-3.7/gnulib-tests'
make[2]: Entering directory '/home/june/HW1/grep-3.7'
make[2]: Leaving directory '/home/june/HW1/grep-3.7'
make[1]: Leaving directory '/home/june/HW1/grep-3.7'
june@june16:~/HW1$ ls
grep-3.7  grep-3.7.tar.gz  script.py  studentid_testcases.txt
june@june16:~/HW1$ 
```

2. Make test cases

open studentid_testcases.txt and write test cases one per line.

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

3. Check the total coverage of your all test-cases

```
$ cd ~/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

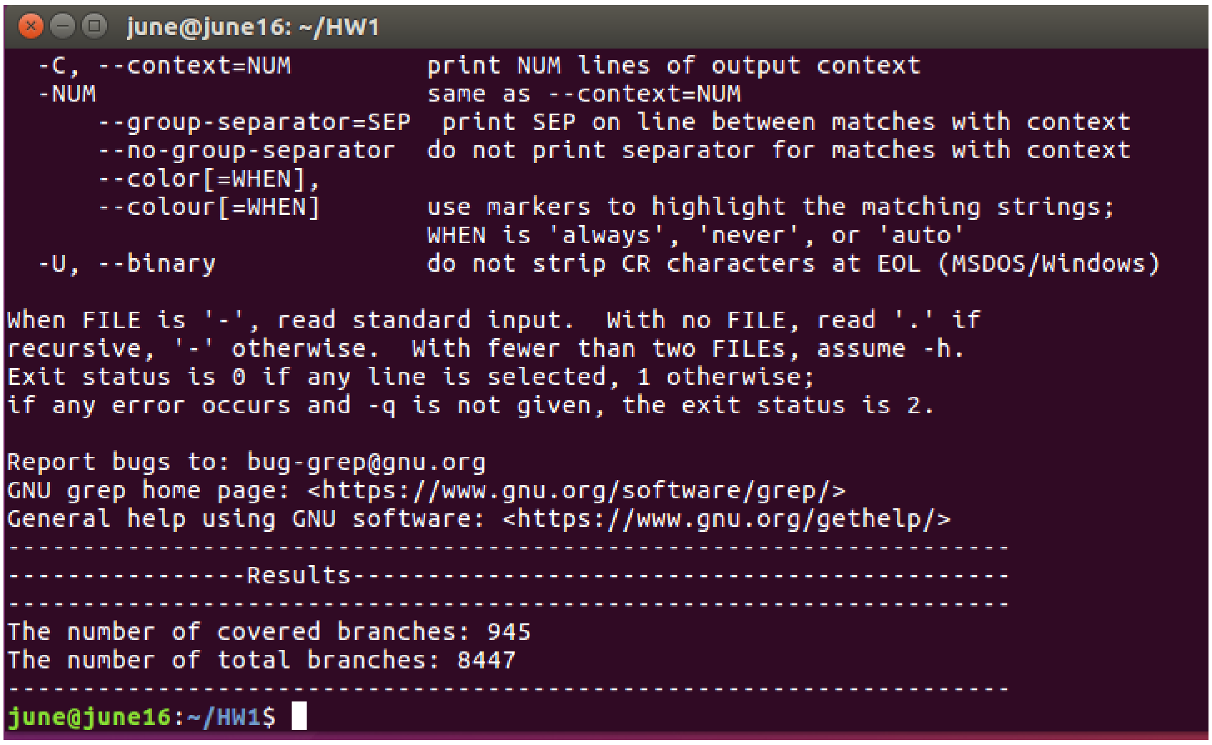
(testcase : command that executes the *grep* binary file)

```
ex) ./grep --help
```

2. After a binary file is executed (by each test-case), *.gcda* files are automatically created for source codes in ***HW1/grep-3.7/src***
   (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



june@june16: ~/HW1

```
  -C, --context=NUM         print NUM lines of output context
  -NUM                      same as --context=NUM
      --group-separator=SEP  print SEP on line between matches with context
      --no-group-separator  do not print separator for matches with context
      --color[=WHEN],
      --colour[=WHEN]       use markers to highlight the matching strings;
                            WHEN is 'always', 'never', or 'auto'
  -U, --binary              do not strip CR characters at EOL (MSDOS/Windows)

When FILE is '-', read standard input.  With no FILE, read '.' if
recursive, '-' otherwise.  With fewer than two FILEs, assume -h.
Exit status is 0 if any line is selected, 1 otherwise;
if any error occurs and -q is not given, the exit status is 2.

Report bugs to: bug-grep@gnu.org
GNU grep home page: <https://www.gnu.org/software/grep/>
General help using GNU software: <https://www.gnu.org/gethelp/>
-------------------------------------------------------------------
----------------Results--------------------------------------------
-------------------------------------------------------------------
The number of covered branches: 945
The number of total branches: 8447
-------------------------------------------------------------------
june@june16:~/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 test-case

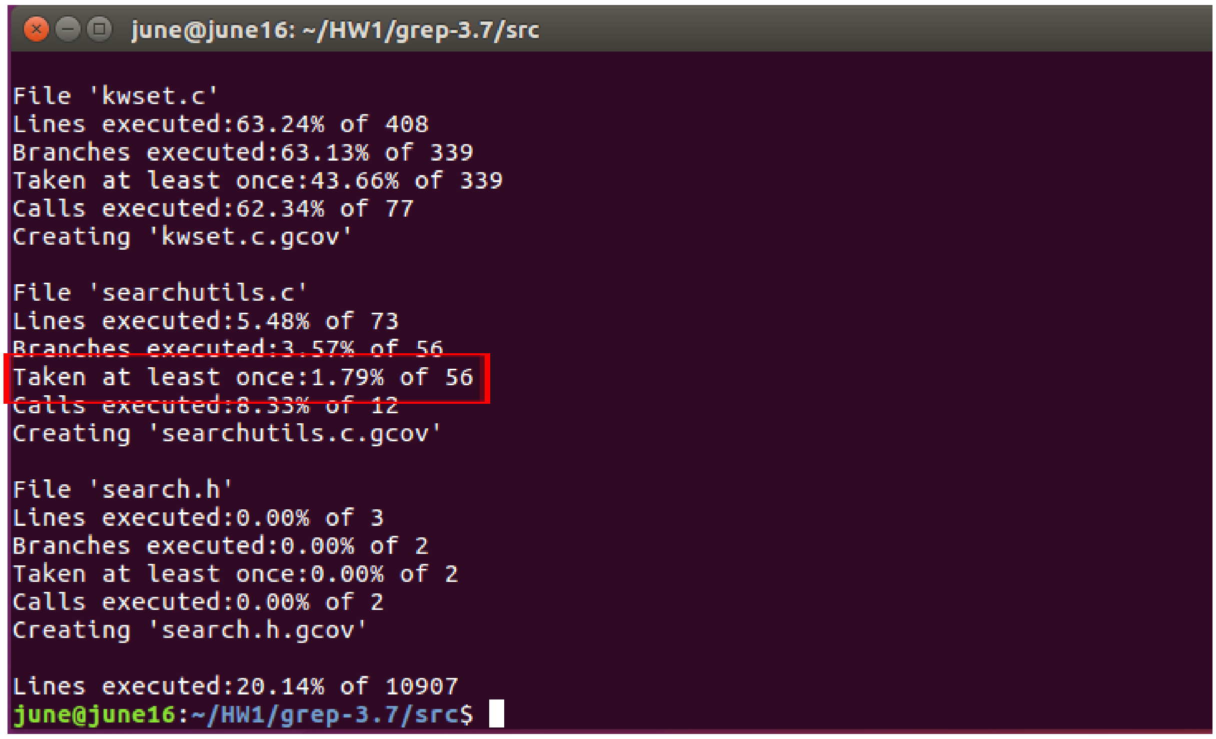
```
$ cd ~/HW1/grep-3.7/src
$ ./grep --help
$ gcov -b ../*/*.gcda
```

Through this, you can check the number of covered branches for each test-case

- In this case, `./grep -help` is the test-case (command)
- That command (test-case) creates the *.gcda* file for each source codes in

HW1/grep-3.7/src directory

- gcov -b ../*/*.gcda will check the number of covered branches by creating the *.gcov* files from *.gcda* files



```
june@june16: ~/HW1/grep-3.7/src

File 'kwset.c'
Lines executed:63.24% of 408
Branches executed:63.13% of 339
Taken at least once:43.66% of 339
Calls executed:62.34% of 77
Creating 'kwset.c.gcov'

File 'searchutils.c'
Lines executed:5.48% of 73
Branches executed:3.57% of 56
Taken at least once:1.79% of 56
Calls executed:8.33% of 12
Creating 'searchutils.c.gcov'

File 'search.h'
Lines executed:0.00% of 3
Branches executed:0.00% of 2
Taken at least once:0.00% of 2
Calls executed:0.00% of 2
Creating 'search.h.gcov'

Lines executed:20.14% of 10907
june@june16:~/HW1/grep-3.7/src$
```

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

For each file,

Taken at least once : 1.79%(<- Covered branch ratio) of 56(<- The number of total branches)

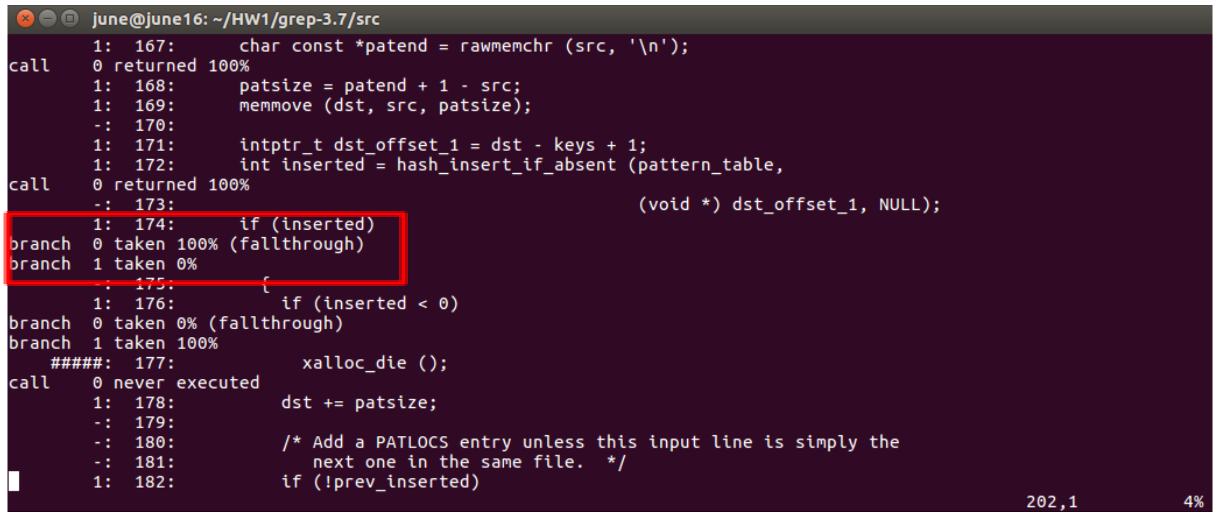
*The number of covered branches= 0.0179 * 56 = 1*

-> 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 test-case

```
$ cd ~/HW1/grep-3.7/src
$ ./grep --help
$ gcov -b ../*/*.gcda
$ vi grep.c.gcov
```

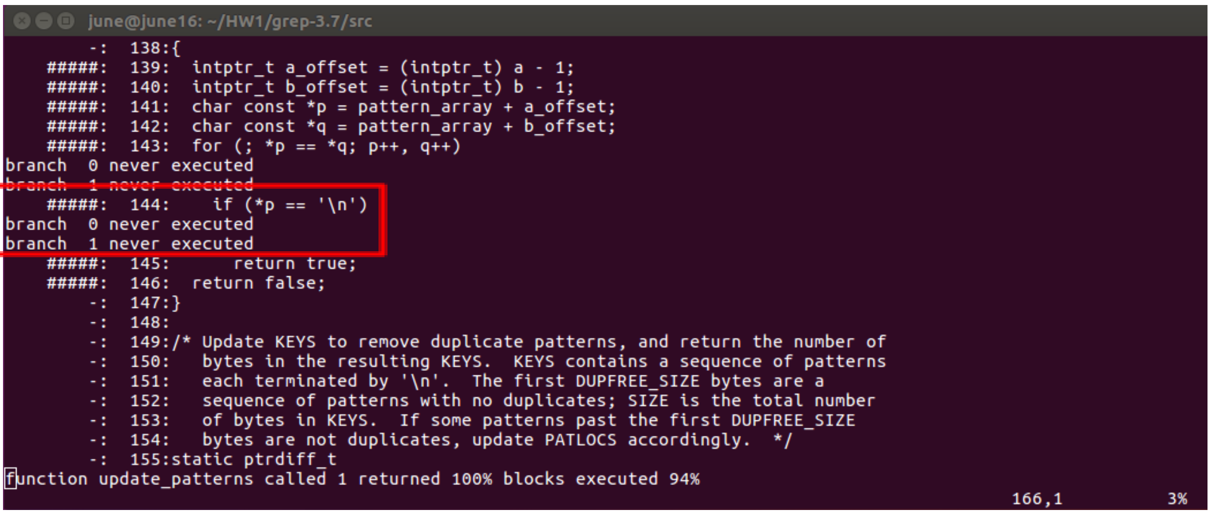
- command creates *.gcda* file
- gcov creates *.gcov* file, which you can read by “vi” or “cat” command
- *.gcov* file shows you which branch is covered
- you can search “branch” keyword in *.gcov* file



```
june@june16: ~/HW1/grep-3.7/src
        1:  167:      char const *patend = rawmemchr (src, '\n');
call    0 returned 100%
        1:  168:      patsize = patend + 1 - src;
        1:  169:      memmove (dst, src, patsize);
        -:  170:
        1:  171:      intptr_t dst_offset_1 = dst - keys + 1;
        1:  172:      int inserted = hash_insert_if_absent (pattern_table,
call    0 returned 100%
        -:  173:                                             (void *) dst_offset_1, NULL);
        1:  174:      if (inserted)
branch  0 taken 100% (fallthrough)
branch  1 taken 0%
        -:  175:        {
        1:  176:          if (inserted < 0)
branch  0 taken 0% (fallthrough)
branch  1 taken 100%
    #####:  177:            xalloc_die ();
call    0 never executed
        1:  178:          dst += patsize;
        -:  179:
        -:  180:          /* Add a PATLOCS entry unless this input line is simply the
        -:  181:             next one in the same file.  */
        1:  182:          if (!prev_inserted)
                                                                          202,1          4%
```

In line 174: if(inserted) -> (in *line 174 in grep.c*, if statement's branch 0, branch 1)

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



```
june@june16: ~/HW1/grep-3.7/src
        -:  138:{
    #####:  139:  intptr_t a_offset = (intptr_t) a - 1;
    #####:  140:  intptr_t b_offset = (intptr_t) b - 1;
    #####:  141:  char const *p = pattern_array + a_offset;
    #####:  142:  char const *q = pattern_array + b_offset;
    #####:  143:  for (; *p == *q; p++, q++)
branch  0 never executed
branch  1 never executed
    #####:  144:    if (*p == '\n')
branch  0 never executed
branch  1 never executed
    #####:  145:      return true;
    #####:  146:  return false;
        -:  147:}
        -:  148:
        -:  149:/* Update KEYS to remove duplicate patterns, and return the number of
        -:  150:   bytes in the resulting KEYS.  KEYS contains a sequence of patterns
        -:  151:   each terminated by '\n'.  The first DUPFREE_SIZE bytes are a
        -:  152:   sequence of patterns with no duplicates; SIZE is the total number
        -:  153:   of bytes in KEYS.  If some patterns past the first DUPFREE_SIZE
        -:  154:   bytes are not duplicates, update PATLOCS accordingly.  */
        -:  155:static ptrdiff_t
function update_patterns called 1 returned 100% blocks executed 94%
                                                                          166,1          3%
```

In line 144: if (*p == '\n') (in *line 144 in grep.c*, 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