Coverage measurement

Purpose of this lesson

- Gather practical experience on how to achieve coverage
- Learn how to use gcov

Gcov

- Open source and free tool from the gnu foundation
- gcov is a test coverage program.
- It is meant to be used in concert with gcc to analyze programs and discover untested parts of it.
- gcov can also be used as a profiling tool to discover where optimization efforts will best affect the code (assess which parts of your code use the greatest amount of computing time). In this case it should be used along with the profiling tool gprof
- Using gcov it is possible to gather information on:
  - how often each line of code executes
  - what lines of code are actually executed
  - how much computing time each section of code uses

Gcov

- Gcov requires that the code is compiled without optimization, because optimization may combine some lines of code into one function.
- gcov accumulates statistics by line (at the lowest resolution), and works best with a programming style that places only one statement on each line.
  - In case of complicated macros that expand to loops or to other control structures, the statistics are less helpful—they only report on the line where the macro call appears.
- gcov creates a logfile called *sourcefile*.gcov which indicates how many times each line of a source file *sourcefile*.c has executed.
- gprof provides additional timing information to use along with the information from gcov.
- gcov works only on code compiled with GCC. It is not compatible with any other profiling or test coverage mechanism.

Invoking gcov

`gcov [`*`options`*`] `*`sourcefiles`*

Options

-h --help

-v --version

-n --no-output

-a --all-blocks

Write individual execution counts for every basic block. Normally gcov outputs execution counts only for the main blocks of a line. With this option you can determine if blocks within a single line are not being executed.

-b --branch-probabilities

Write branch frequencies to the output file, and branch summary info to the standard output. Allows to see how often each branch is taken. Unconditional branches are not shown, unless the -u option is given.

Invoking gcov

-c --branch-counts

Write branch frequencies as the number of branches taken, rather than the percentage of branches taken.

--long-file-names

Create long file names for included source files. For example, if the header file x.h contains code, and was included in a.c, then running gcov on the file a.c will produce an output file called a.c##x.h.gcov instead of x.h.gcov. This can be useful if x.h is included in multiple source files. If you use the `-p' option, both the including and included file names will be complete path names.

-p --preserve-paths

Preserve complete path information in the names of generated .gcov files. Without this option, just the filename component is used. With this option, all directories are used, with `/' characters translated to `#' characters, . directory components removed and .. components renamed to `^'. This is useful if sourcefiles are in several different directories. It also affects the `-l' option.

Invoking gcov

-f --function-summaries

Output summaries for each function in addition to the file level summary.

-o *directory|file*

--object-directory *directory*

--object-file *file*

Specify either the directory containing the gcov data files, or the object path name. The .gcno, and .gcda data files are searched for using this option. If a directory is specified, the data files are in that directory and named after the source file name, without its extension. If a file is specified here, the data files are named after that file, without its extension. If this option is not supplied, it defaults to the current directory.

-u --unconditional-branches

When branch probabilities are given, include those of unconditional branches. Unconditional branches are normally not interesting.

Invoking gcov

- To allow finding the source files, gcov should be run with the current directory set as the same from which the compiler is called.
- gcov produces one coverage information file (extension .gcov) for each source file of the program the current directory. The *name* part of the output file name is usually simply the source file name, but can be more complicated if the `-l' or `-p' options are given.
- When using gcov, the program must be compiled with the GCC options: `-fprofile-arcs -ftest-coverage'. These tell the compiler to generate additional information needed by gcov (basically a flow graph of the program) and also includes additional code in the object files for generating the extra profiling information needed by gcov. These additional files are placed in the directory where the object file is located.
- Running the program will cause profile output to be generated. For each source file compiled with -fprofile-arcs, an accompanying .gcda file will be placed in the object file directory.

Invoking gcov

The .gcov files contain the `:' separated fields along with the program source code

execution_count:*line_number*:*source line text*

Additional block information may follow each line, when requested by command line options. The *execution_count* is `-' for lines containing no code and `#####' for lines which were never executed.

The preamble lines are of the form

-:0:*tag*:*value*

The *tag* name should be used to locate a preamble line.

The additional block information is of the form

tag information

The *information* is human readable, and simple enough for machine parsing.

When printing percentages, 0% and 100% are only printed when the values are *exactly* 0% and 100%. Other values which would conventionally be rounded to 0% or 100% are printed as the nearest non-boundary value.

Invoking gcov

- Running gcov with your program's source file names as arguments will now produce a listing of the code along with frequency of execution for each line. For example, if your program is called tmp.c, this is what you see when you use the basic gcov facility:

```
$ gcc -fprofile-arcs -ftest-coverage
  tmp.c
$ a.out
$ gcov tmp.c
90.00% of 10 source lines executed in
  file tmp.c
Creating tmp.c.gcov.
```

Invoking gcov

The file tmp.c.gcov contains output from gcov. Here is a sample:

```
    -:  0:Source:tmp.c
    -:  0:Graph:tmp.gcno
    -:  0:Data:tmp.gcda
    -:  0:Runs:1
    -:  0:Programs:1
    -:  1:#include <stdio.h>
    -:  2:
    -:  3:int main (void)
    1:  4:{
    1:  5: int i, total;
    -:  6:
    1:  7: total = 0;
    -:  8:
   11:  9: for (i = 0; i < 10; i++)
   10: 10: total += i;
    -: 11:
    1: 12: if (total != 45)
#####: 13: printf ("Failure\n");
    -: 14: else
    1: 15: printf ("Success\n");
    1: 16: return 0;
    -: 17:}
```

Invoking gcov

When you use the -a option, you will get individual block counts, and the output looks like this:

```
...
    -:  0:Programs:1
    -:  1:#include <stdio.h>
    -:  2:
    -:  3:int main (void)
    1:  4:{
    1:  4-block  0
    1:  5: int i, total;
    -:  6:
    1:  7: total = 0;
    -:  8:
   11:  9: for (i = 0; i < 10; i++)
   11:  9-block  0
   10: 10: total += i;
   10: 10-block  0
    -: 11:
    1: 12: if (total != 45)
    1: 12-block  0
#####: 13: printf ("Failure\n");
$$$$$: 13-block  0
    -: 14: else
    1: 15: printf ("Success\n");
    1: 15-block  0
    1: 16: return 0;
    1: 16-block  0
    -: 17:}
```

Invoking gcov

- In this mode, each basic block is only shown on one line – the last line of the block. A multi-line block will only contribute to the execution count of that last line, and other lines will not be shown to contain code, unless previous blocks end on those lines. The total execution count of a line is shown and subsequent lines show the execution counts for individual blocks that end on that line. After each block, the branch and call counts of the block will be shown, if the -b option is given.
- Because of the way GCC instruments calls, a call count can be shown after a line with no individual blocks. As you can see, line 13 contains a basic block that was not executed.

Invoking gcov

When you use the -b option, your output looks like this:

```
$ gcov -b tmp.c
90.00% of 10 source lines executed in file tmp.c
80.00% of 5 branches executed in file tmp.c
80.00% of 5 branches taken at least once in file tmp.c
50.00% of 2 calls executed in file tmp.c
Creating tmp.c.gcov.
```

Invoking gcov

Here is a sample of a resulting tmp.c.gcov file:

```
-:  0:Source:tmp.c
-:  0:Graph:tmp.gcno
-:  0:Data:tmp.gcda
-:  0:Runs:1
-:  0:Programs:1
-:  1:#include <stdio.h>
-:  2:
-:  3:int main (void)
function main called 1 returned 1 blocks executed 75%
1:  4:{
1:  5:  int i, total;
-:  6:
1:  7:  total = 0;
-:  8:
11:  9:  for (i = 0; i < 10; i++)
branch 0 taken 91% (fallthrough)
branch 1 taken 9%
```

For each function, a line is printed showing how many times the function is called, how many times it returns and what percentage of the function's blocks were executed.

For a branch, a percentage indicating the number of times the branch was taken is printed if it is executed at least once. Otherwise, the message “never executed”.

Invoking gcov

```
   10: 10: total += i;
    -: 11:
    1: 12: if (total != 45)
branch 0 taken 0% (fallthrough)
branch 1 taken 100%
#####: 13: printf ("Failure\n");
call 0 never executed
    -: 14: else
    1: 15: printf ("Success\n");
call 0 called 1 returned 100%
    1: 16: return 0;
    -: 17:}
```

For a call, a percentage indicating the number of times the call returned is shown, if it is executed at least once.

- This is usually 100%, but may be less for functions that call exit or longjmp

For each basic block, a line is printed after the last line of the basic block describing the branch or call that ends the basic block.

- There can be multiple branches and calls listed for a single source line if there are multiple basic blocks that end on that line.

Invoking gcov

- The execution counts are cumulative.
- If the program is executed again without removing the .gcda file, the count for the number of times each line in the source is executed are added to the results of the previous run(s).
- This is useful in several ways. For example, it can be used to accumulate data over a number of program runs as part of a test verification suite
- The data in the .gcda files is saved immediately before the program exits.
- For each source file compiled with -fprofile-arcs, the profiling code first attempts to read in an existing .gcda file; if the file doesn't match the executable (differing number of basic block counts) it will ignore the contents of the file. It then adds in the new execution counts and finally writes the data to the file.