**Code coverage practice session**

Typographical conventions

We use the following conventions in this guide:

emacs The name of a specific command or file

file You should replace file with a specific name

Exit abc Output that you see on the screen

Getting Started with gcov

1. Login into the Linux server with your login Ids

A screenshot of a computer screen

Description automatically generated

2. Create a new directory called code\_cov in your home directory <home>

Ans: mkdir code\_cov

3. Go inside the directory you have created in (2) /<home>/code\_cov

Ans: cd code\_cov

4. Copy the following files from the path as mentioned by the trainer:

a. sample.c

b. link.c

c. link.h

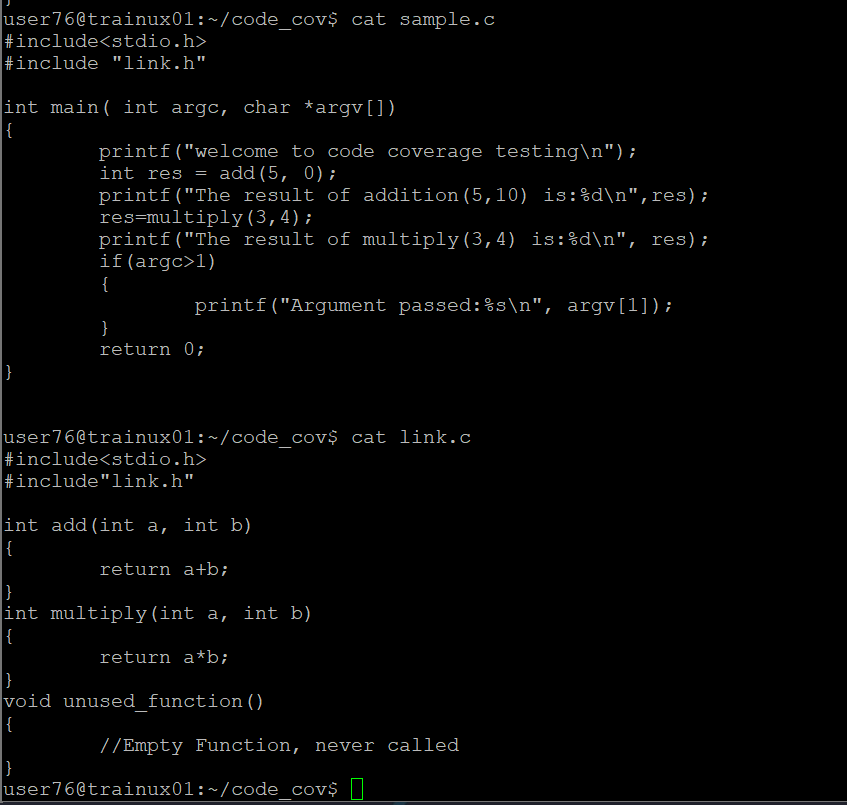
Ans:

A screenshot of a computer program

Description automatically generated

5. Take a look at the example programs sample.c and link.c

Compilation



6. Compile the files sample.c and sample1.c and put the output in the executable file called output

Ans: gcc –o output –ftest-coverage –fprofile-arcs sample.c link.c

The .gcno file is generated when the source file is compiled with the GCC -ftest-coverage option and .gcda file is generated when a program containing object files built with the GCC -fprofile-arcs option is executed



7. Execute the file output

Ans: ./output

8. Now run gcov for each source file one by one

Ans: gcov sample.c

File `sample.c'

Lines executed:50.00% of 10

sample.c:creating `sample.c.gcov'

View the output file sample.c.gcov with the vi editor

vi sample.c.gcov

Analyse this file and notice that each executable statement is either preceeded by a number or by ######. The number specifies the number of times that statement got executed while ###### represents that this source code statement did not get executed at all.

gcov link.c

File `link.c'

Lines executed:100.00% of 2

sample1.c:creating `link.c.gcov'

9. Run output again, this time with command line arguments:

./output a a b b

Check!

oops

Check!

This function is just called to link this file

10. Now run gcov for sample.c again

What do you observe?

NOTE : If the code coverage is not 100% , it can be achieved by using gdb (For that, you need to compile with -g as well as -ftest-coverage -fprofile-arcs options)

Food for thought : What happens to the currently achieved coverage, when you modify a .c file? Is the earlier coverage data still valid?