Consider a simple code below and do the experiment as suggested. Here we will be using different types of tools to analyze and identify the errors.

We will be using the following tools during Coding and Unit testing:

1. Static Code Analyzer : Klocwork , splint ,lint etc
2. Dynamic Analysis :
   1. Debugger: gdb
   2. Memory leak : valgrind ,purify
   3. Profiling :gprof
3. Unit Testing :
   1. Code Coverage : gcov

Example1.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define SIZE 16

int main()

{

char \*data1, \*data2, \*tmp1, \*tmp2;

data1 = (char \*)malloc(SIZE);

if (NULL == data1)

exit (EXIT\_FAILURE);

printf("Please input your username: ");

if (EOF == scanf("%s", data1))

exit (EXIT\_FAILURE);

data2 = (char \*)malloc (SIZE);

if (NULL == data2)

exit (EXIT\_FAILURE);

tmp1 = data1;

tmp2 = data2;

while (\*tmp1 != '\0')

{

\*tmp2 = \*tmp1;

tmp1++;

tmp2++;

}

tmp1 = data1;

free (data1);

printf ("data2 :%s:\n", data2);

/\*Klocwork is able to catch the fact that memory pointed to by tmp is already

freed at line no 37 and is being read in the following instruction at line no 44\*/

printf ("tmp1 :%s:\n", tmp1);

/\* Klocwork is able to catch the fact that the memory allocated for data2 at line no 39 is not freed .

\*/

return 0;

}

1. **Follow the given steps to do the static code analysis using Klocwork.(You can use any static code analysis tool for reviewing your code). Refer to the PPT attached with the demo document to understand the usage of Klocwork.**

server commands:

export SDEDIR=/osp/sde/L\_n\_D/src/sample\_c/

export PATH=$PATH:/osp/sde/Klocwork\_U\_lic/bin

export MKDOCLEAN="yes"

export MKDODEP="yes"

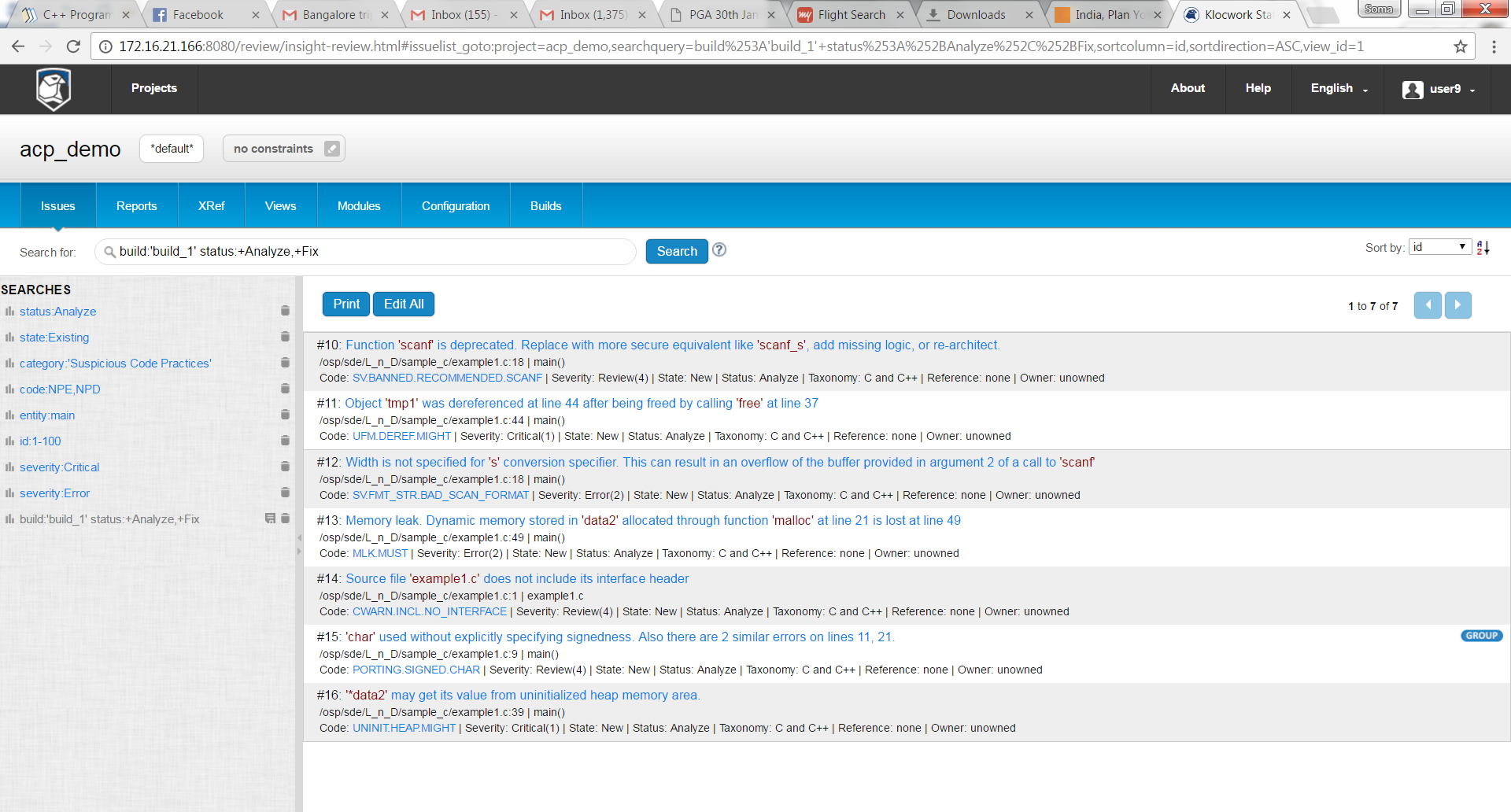
make clean

$>kwinject -o example1.out makefile1

$> kwbuildproject --url http://172.16.21.166:8080/acp\_demo -o example1\_tables example1.out

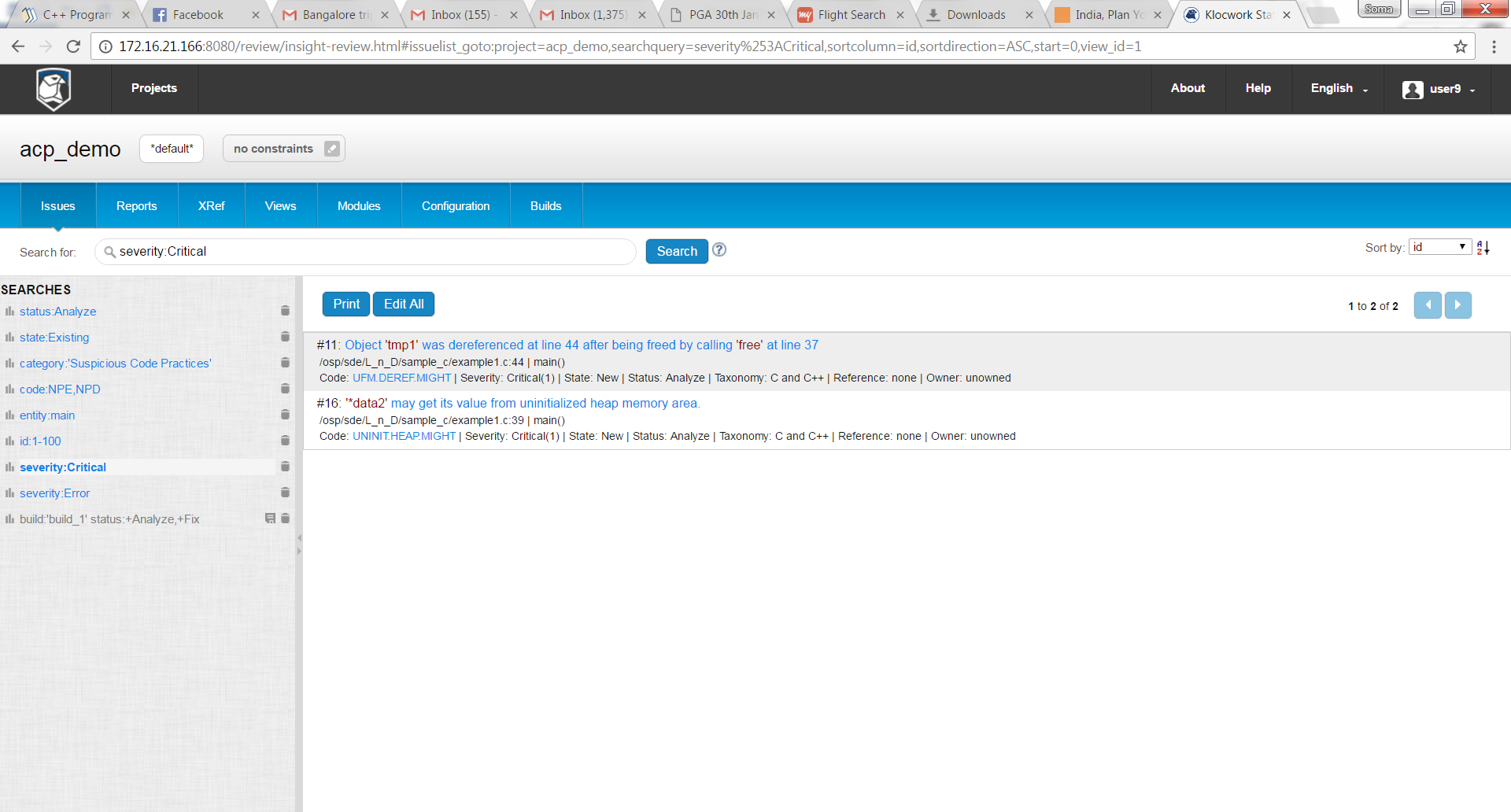
$ kwadmin --url http://172.16.21.166:8080 load acp\_demo example1\_tables/

Let us now try to figure out, what is identified by the static code analyzer by analyzing the output.



Conclusion:If you observe the output there are different severity of errors i.e. critical, error etc listed by the tool.(Refer to the severity levels in the presentation attached)

Following are the critical errors which needs to be fixed .



**Observation from Static code analysis:**

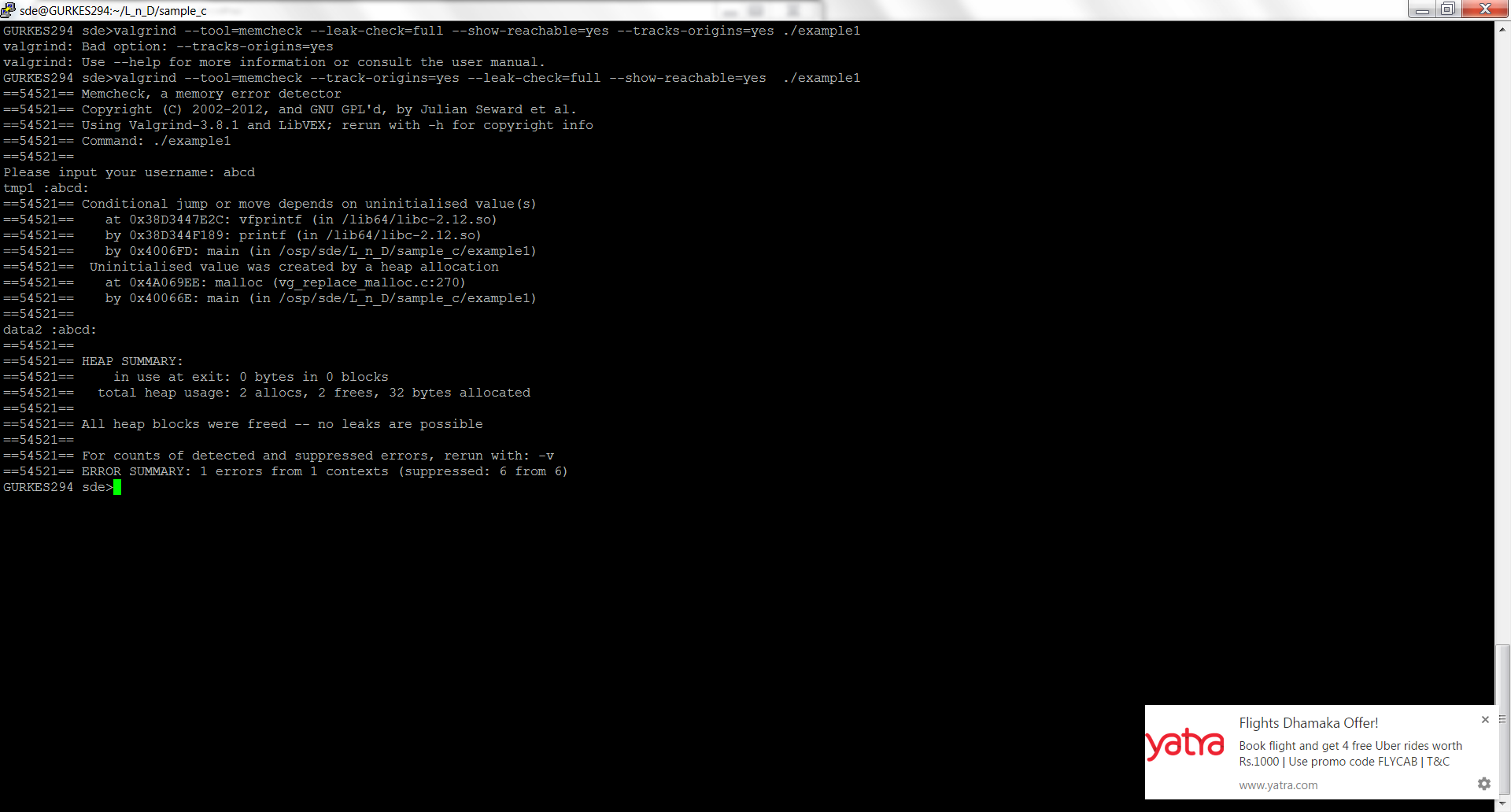
1. Klocwork is able to catch the fact that memory pointed to by tmp is already freed at line no 37 and is being read in the following instruction at line no 44

2. Klocwork is able to catch the fact that the memory allocated for data2 at line no 39 is not freed .

1. **Consider a Memory leak tool ie. Valgrind, purify etc on the same code. Follow the below commands to compile and execute the code :You can refer to the attached demo doc to understand the output.**

$>gcc example1.c

$> valgrind --tool=memcheck --leak-check=full --show-reachable=yes ./example1



Observation: Since all the heap blocks are free so no memory leak .

1. **Code Coverage tool:After fixing the errors using the static(Klocwork) and dynamic analysis(gdb,valgrind) tool ,we try to ensure the coverage of the code is 100% using gcov**

Consider the same code and do the following suggested steps to see the code coverage.Refer to the Demo document and the PPT provided as an attachment.

Follow the below steps

1. gcc -o example1 -ftest-coverage -fprofile-arcs -g example1.c
2. ./example1
3. gcov example1.c

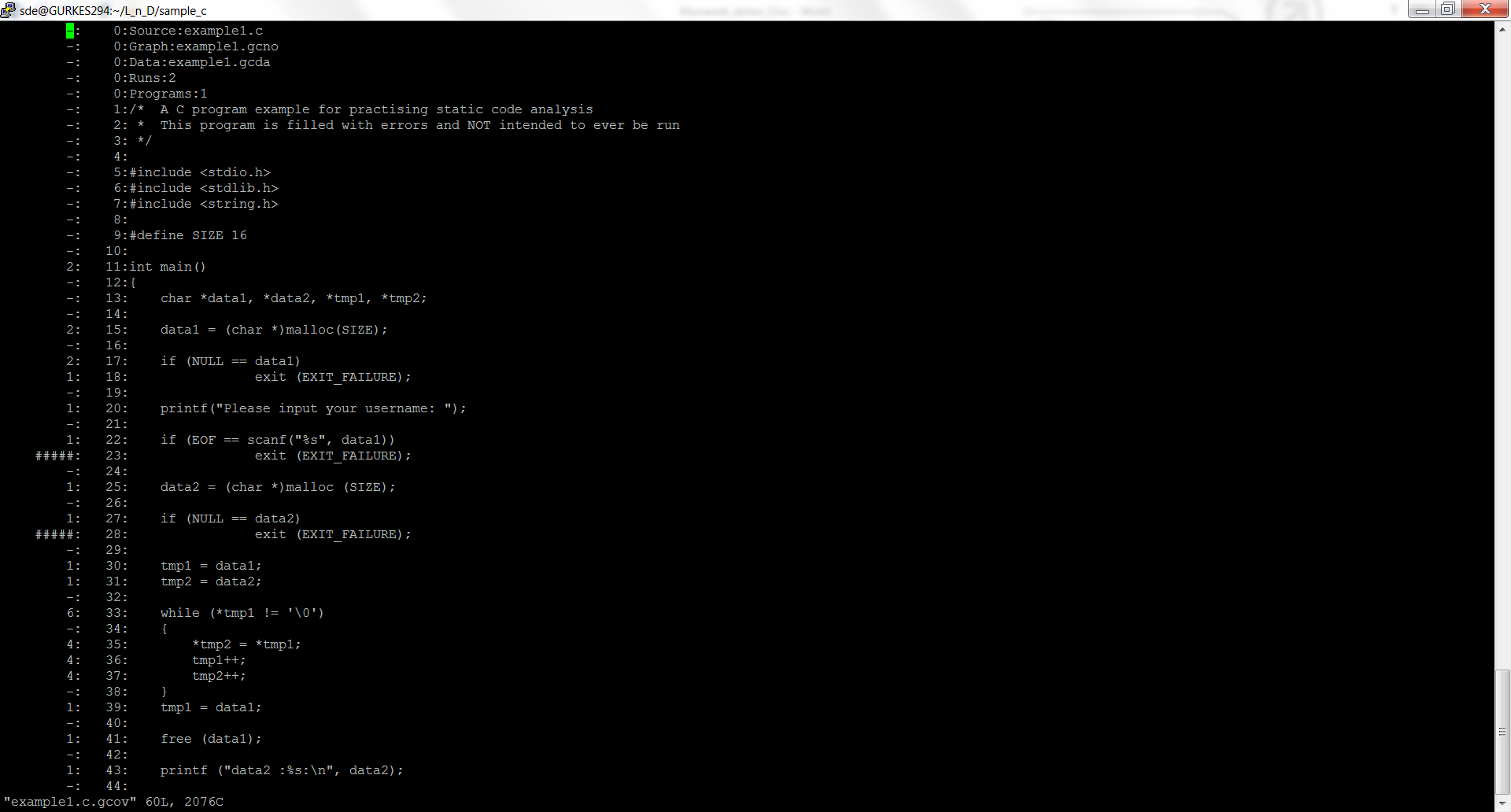
following is an output:

File 'example1.c'

Lines executed:90.48% of 21

example1.c:creating 'example1.c.gcov'

It indicates that the code coverage is 90.48% and the code coverage need to be enhanced to 100% to ensure that all the lines of code executes.

1. vi example1.c.gcov

##### :It indicates that the following lines did not execute .So the following lines need to be executed to ensure that code is 100% covered.