Installation samt exempelanvändning av Google Test i Linux

1. Installera Google Test framework:
   $ sudo apt install libgtest-dev

2. Installera cmake om du inte redan har gjort detta:
   $ sudo apt install cmake

3. Kompilera biblioteket libgtest:
   $ cd /usr/src/gtest
   $ sudo cmake CMakeLists.txt
   $ sudo make

4. Kopiera filer libgtest.a och libgtest_main.a till katalogen /usr/lib/folder:
   $ cd lib
   $ sudo cp *.a /usr/lib

5. Skapa en ny katalog döpt *gtest_example.* I denna katalog, skapa en fil döpt *CMakeLists.txt* och lägg till följande:

```
cmake_minimum_required(VERSION 3.20)
project(gtest_example)
find_package(GTest REQUIRED)
include_directories(${GTEST_INCLUDE_DIRS})
add_executable(run_unit_test unit_test.cpp)
target_compile_options(run_unit_test PRIVATE -Wall -Werror)
target_link_libraries(run_unit_test ${GTEST_LIBRARIES} pthread)
```

   Ovanstående kommandon medför att en körbar fil döpt *run_unit_test* kommer skapas, där en enda källkodsfil *unit_test.cpp* innehåller testrutinerna samt all kod som ska testas. Övriga kommandon kan läggas till i vanlig ordning, exempelvis *include_directories* för att inkludera undermapp innehållande headerfiler.

6. Skapa tidigare nämnd fil *unit_test.cpp* för unit testing och lägg till följande inkluderingsdirektiv:

```
#include <gtest/gtest.h>
```

7. Lägg till följande main-funktion för att köra alla tester:

```
int main(int argc, char** argv) {
    testing::InitGoogleTest(&argc, argv);
    return RUN_ALL_TESTS();
}
```

8. Tester definieras via den makroliknande funktionen TEST, såsom visas nedan:

```
TEST(<Testets titel>, <Specifikt vad som testas) {

}
```

   För att genomföra tester används olika typer av assertion-funktioner *(assertion = påstående),* exempelvis EXPECT_EQ samt EXPECT_NEAR, se följande länk: http://google.github.io/googletest/reference/assertions.html

   Se bifogad fil *unit_test.cpp* för exempel på enhetstestning av C-funktioner för addition och subtraktion.
   OBS! Filens innehåll visas även i bilaga B nedan.

9. Skriv följande kommandon i terminalen för att kompilera och köra enhetstestet:
$ cmake CMakeLists.txt
$ make
$ ./run_unit_test

Bilaga B – Filen *unit_test.cpp* för enhetstestning av C-funktioner

```
/*****************************************************************************
 * @brief Example of unit testing of C functions for addition and division.
 *****************************************************************************/
#include <gtest/gtest.h>

/*****************************************************************************
 * @brief C code to be tested.
 *
 * @note extern "C" is used to compile code with the C compiler.
 *       extern "C" can be placed in C headers to make it possibly to use
 *       C code in C++ programs.
 *****************************************************************************/
extern "C" {

/*****************************************************************************
 * @brief Adds two integers.
 *
 * @param x
 *        The first integer to add.
 * @param y
 *        The second integer to add.
 * @return
 *        The sum of the two integers.
 *****************************************************************************/
static inline int add_int(const int x, const int y) {
    return x + y;
}

/*****************************************************************************
 * @brief Divides two integers.
 *
 * @param x
 *        The dividend.
 * @param y
 *        The divisor (must not be 0).
 * @return
 *        The quotient of the floating-point numbers or 0 if the divisor is 0.
 *****************************************************************************/
static inline double divide_int(const int x, const int y) {
    return y != 0 ? x / (double)(y) : 0;
}
```

```
/**************************************************************************
 * @brief Adds two floating-point numbers.
 *
 * @param x
 *        The first floating point number to add.
 * @param y
 *        The second floating number to add.
 * @return
 *        The sum of the two floating-point numbers.
 *************************************************************************/
static inline double add_double(const double x, const double y) {
    return x + y;
}

/**************************************************************************
 * @brief Divides two floating-point numbers.
 *
 * @param x
 *        The dividend.
 * @param y
 *        The divisor (must not be 0).
 * @return
 *        The quotient of the floating-point numbers or 0 if the divisor is 0.
 *************************************************************************/
static inline double divide_double(const double x, const double y) {
    return y != 0 ? x / y : 0;
}
} /* extern "C" */

/**************************************************************************
 * @brief Tests the addition functions. When performing floating-point addition,
 *        the precision of the sum is set to 0.001.
 *************************************************************************/
TEST(MathTest, AdditionTest) {
    EXPECT_EQ(add_int(3, 4), 7);
    EXPECT_EQ(add_int(3, -4), -1);
    EXPECT_NEAR(add_double(3.5, 4.2), 7.7, 0.001);
    EXPECT_NEAR(add_double(3.5, -4.2), -0.7, 0.001);
}
```

```
/*****************************************************************************
 * @brief Tests the division functions. The precision of the quotient is set
 *        to 0.001. If an invalid divisor is passed (0), all division functions
 *        shall return exactly 0.
 *****************************************************************************/
TEST(MathTest, DivisionTest) {
    EXPECT_NEAR(divide_int(3, 4), 0.75, 0.001);
    EXPECT_NEAR(divide_int(3, -4), -0.75, 0.001);
    EXPECT_EQ(divide_int(1, 0), 0);

    EXPECT_NEAR(divide_double(3.2, 2), 1.6, 0.001);
    EXPECT_NEAR(divide_double(3.2, -2), -1.6, 0.001);
    EXPECT_EQ(divide_double(1, 0), 0);
}

/*****************************************************************************
 * @brief Runs all unit tests.
 *
 * @param argc
 *        The number of input arguments entered in the terminal when running
 *        the program.
 * @param argv
 *        Reference to vector containing all the input arguments entered in
 *        the terminal.
 * @return
 *        Success code 0 if all tests could be run, else error code 1.
 *****************************************************************************/
int main(int argc, char** argv) {
    testing::InitGoogleTest(&argc, argv);
    return RUN_ALL_TESTS();
}
```