Programación Orientada a Objetos. Curso 2023-24
PRÁCTICA 3. El proyecto *marketplace*. Google Test

1.- Google Test

Los *tests* son imprescindibles en programación. No solo permiten asegurar que nuestro código funciona, sino que son de gran ayuda para el desarrollo en lo que se conoce como: desarrollo guiado por tests o test-driven development (TDD). Esto es así independientemente de la metodología de programación y del lenguaje de programación que usemos.
Nosotros vamos a ir haciendo tests de cada unidad o módulo de software (lo que se conoce como unit testing o pruebas unitarias) que vayamos codificando en C++ y para ello vamos a usar *Google Test*.
Google Test es uno de los frameworks de Unit Testing más utilizados. También se puede considerar un framework de tipo 'xUnit', que es una familia de frameworks para pruebas unitarias que comparten características comunes.

2.- Estructura de nuestro proyecto

En esta práctica vamos a comenzar el desarrollo de un nuevo proyecto al que vamos a denominar 'marketplace'. Por tanto crea el directorio poo/p3/marketplace.
En este directorio tendremos 3 directorios principales:

- marketplace/build con los ejecutables.
- marketplace/src con el código fuente.
- marketplace/tests con los tests.

Crea cada uno de estos directorios. La estructura de este proyecto irá quedando como se muestra en la Figura 1.

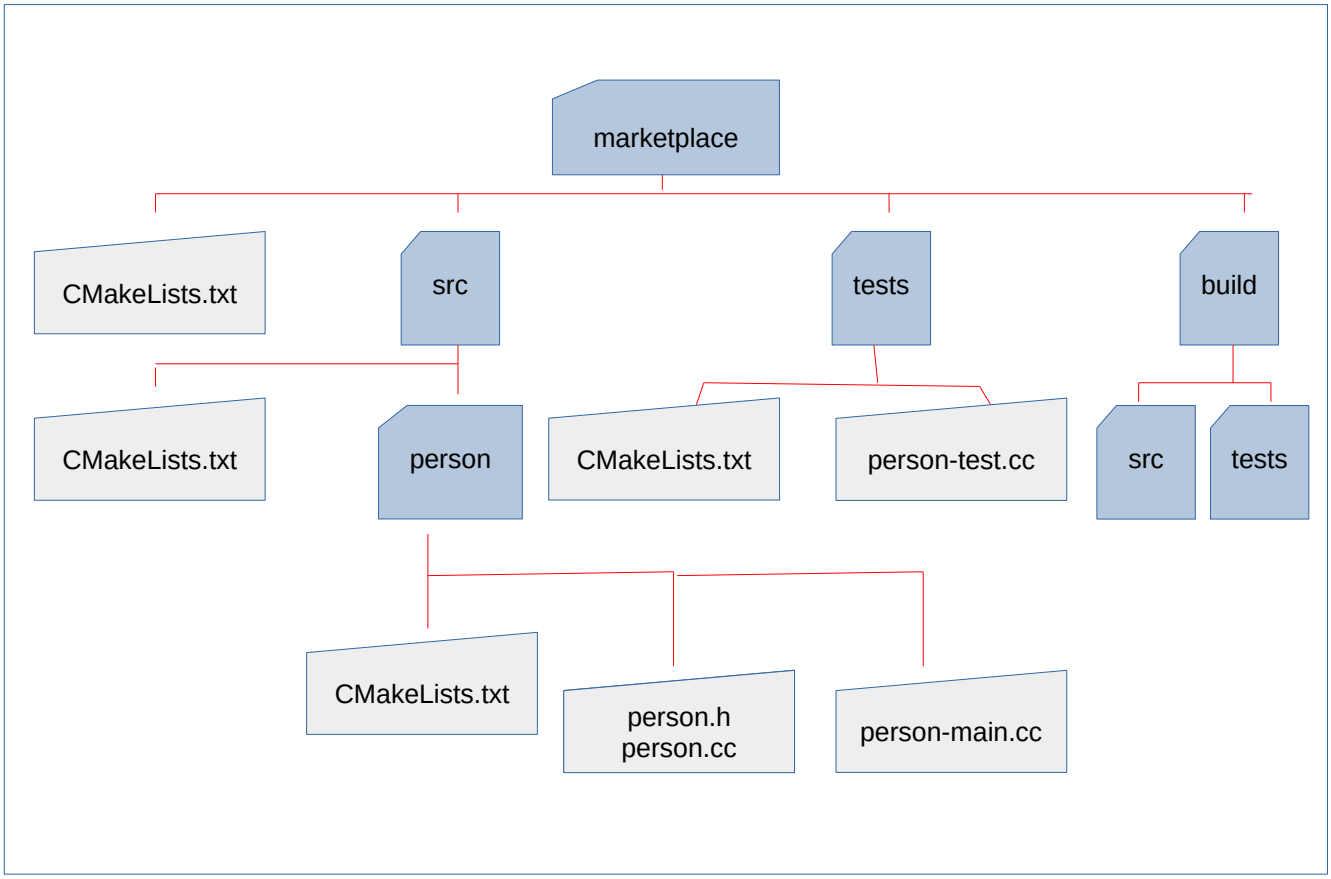

Figura 1: Estructura de directorios del proyecto.

Copia el top-level CMakeLists.txt del proyecto de la anterior práctica que se encuentra en poo/p2/

dentro del directorio poo/p3/marketplace. En este fichero vamos a añadir la creación de tests para nuestro proyecto. Esto se hace insertando una línea antes de la inclusión de subdirectorios, quedando este fichero de la siguiente manera:

```
cmake_minimum_required(VERSION 3.10)

project(marketplace)

set(CMAKE_CXX_STANDARD 17)

set(CMAKE_CXX_STANDARD_REQUIRED True)

# Enables testing with add_test() for this project
enable_testing()

# Adding subdirs
add_subdirectory(src)
add_subdirectory(tests)
```

El CMakeLists.txt del directorio src deberá contener:

```
add_subdirectory(person)
```

Y el CMakeLists.txt del directorio src/person se puede quedar como estaba en la práctica anterior.

Copia person.h y person.cc (no copies el fichero person-main.cc) de la práctica anterior en el directorio:

poo/p3/marketplace/src/person

Descarga de moodle el fichero person-test.cc y copialo en el directorio marketplace/tests/

En el directorio marketplace/tests debe haber un fichero CMakeLists.txt con la siguientes instrucciones (este fichero lo tienes en moodle):

```
# GTest include code for fetch content over the internet
include(FetchContent)

FetchContent_Declare(
  googletest
  GIT_REPOSITORY https://github.com/google/googletest.git
  GIT_TAG        v1.14.0
)
FetchContent_MakeAvailable(googletest)
add_library(GTest::GTest INTERFACE IMPORTED)
target_link_libraries(GTest::GTest INTERFACE gtest_main gmock_main)
# End GTest include code

# Testing person library
add_executable(person-test person-test.cc)
target_link_libraries(person-test
PRIVATE GTest::GTest person)
add_test(person-gtests person-test)
```

La parte entre #GTest y # End GTest se corresponde con la descarga automática de Google Test desde su repositorio en GitHub cada vez que ejecutes CMake.
Del resto de instrucciones, add_executable() ya la hemos visto; target link_libraries() es para linkar person-test con la librería de Google Test (GTest::GTest) y con person (indicamos PRIVATE ya que solo se añaden al target actual).
Ahora ya tenemos nuestro proyecto listo para ejecutar los tests que hay en el fichero person.-test.cc que te has descargado, pero antes, vamos a añadir funcionalidad a la clase Person.

3.- Añadiendo más funcionalidad para la clase Person.
A partir del fichero person.h de la práctica 2, añade los siguientes datos a la clase Person:

- id_ de tipo string que identifica unívocamente a una persona en el marketplace.
- town_: de tipo string con la ciudad
- province_: de tipo string con la provincia
- country_: de tipo string con el país
- entry_year_: de tipo int con el año de alta de la persona en el sistema

Añade la siguientes funciones a la clase Person:

- Constructor que recibe los siguientes parámetros y en este orden: id, name, town, province, country, age, rank y entry_year. El parámetro *id* es obligatorio. El resto de parámetros de tipo std::string tendrán como valor por defecto la cadena “empty”. Los parámetros *age*, *rank* y *entry_year* tendrán como valor por defecto 0. Si en la llamada se proporciona *entry_year* y no es mayor de 2000 se pondrá a 0.
- Observadores/getters GetId(), GetName(), GetTown(), GetProvince(), GetCountry(), GetAge(), GetRank() y GetEntryYear().
- Modificadores/setters: SetId(), SetName(), SetTown(), SetProvince(), SetCountry() y SetRank().
- SetAge() que debe devolver un dato de tipo ***bool*** y que recibe como parámetro un entero con la nueva edad que ser mayor a 0. Si el valor recibido es >0 se cambia el atributo correspondiente y se devuelve ***true***, en caso contrario, no se cambia y se devuelve ***false***.
- SetEntryYear() que debe devolver un dato de tipo ***bool*** y que recibe como parámetro un entero con el nuevo año que tiene que ser mayor a 2000. Si el valor recibido es >2000 se cambia el atributo correspondiente y se devuelve ***true***, en caso contrario, no se cambia y se devuelve ***false***.
- Cambia la función GetDataStr() para que devuelva un string con el siguiente contenido:
  - Una línea inicial con la cadena “Person: \n”
  - Seguida de una línea por cada atributo, en el mismo orden que el indicado en el constructor, que indique: Nombre del Atributo + “: ” + Valor del atributo + “\n”
  - Por ejemplo:
    Person:
    id: 778
    name: Ana
    town: Sevilla
    ….

Vamos a hacer un pequeño programa principal en el fichero *src/*person/person-main.cc que declare dos objetos de tipo Person y que saque en pantalla sus datos con GetDataStr().
Vamos a compilar todo y hacer los tests de todo en el siguiente apartado.

4.- Pasando los tests de la clase Person
Ya te has descargado de moodle el fichero con los tests (person-test.cc) al directorio marketplace/tests.
A continuación vamos a ejecutar ya los tests del fichero person-test.cc a nuestra clase Person ya finalizada:

1. Analiza el contenido del fichero person-test.cc
2. Ejecuta `cmake ..` dentro del directorio ‘build’

3. Ejecuta `make` dentro del directorio ‘build’
4. Ve al directorio ‘build/tests’ y ejecuta el programa ‘person_test’. O bien desde el directorio build ejecuta:
   ```
   $ ./tests/person-test
   ```
   El último test prueba showPreferences() y debe sacar en pantalla lo siguiente:
   futbol
   fashion
   surf
   cinema
   padel
   golf
   rock
   music
5. Analiza cada línea de salida y corrige los posibles errores de tu código.
6. Ve al directorio ‘build’ y ejecuta el programa ‘ctest’. Analiza cada línea de salida.
7. Vamos a utilizar este tipo de test a lo largo de todo el curso. Para familiarizarnos mejor con ellos podemos introducir algún error en cada una de las operaciones de la clase Person y ejecutar los test. Ir corrigiendo cada error, uno a uno y ejecutando los tests de nuevo después de cada corrección, hasta que se corrijan todos los errores.
8. Por último limpia y optimiza el código de toda la clase Person para que quede al máximo nivel de calidad, autodocumentación, estética, claridad, etc. Y vuelve a ejecutar los tests para confirmar que todo ha quedado correcto.
9. Ve al directorio build/src/person y ejecuta el programa person-main que hiciste en el apartado anterior. O bien desde el directorio build ejecuta:

   ```
   $ ./src/person/person-main
   ```

Por último, echa un vistazo y estudia todo el material y todos los enlaces que se han dejado en el moodle de la asignatura correspondientes a esta práctica. Verás que también hay una sección con material adicional y enlaces de interés en el moodle de la asignatura que debes revisar de vez en cuando para familiarizarte con todo ello.