Guion de prácticas

Shopping1

DateTime, Event

Febrero de 2021

UNIVERSIDAD DE GRANADA

ETSIIT Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

DECSAI

Metodología de la Programación

DGIM-GII-GADE

Curso 2020/2021

Índice

1. Descripción

En un sitio web de venta por internet, los usuarios se conectan, navegan por los productos y, eventualmente, compran algunos de ellos añadiéndolos al carrito de la compra.

Estas operaciones de cada usuario dejan un registro en el servidor, el cual recoge datos como la fecha, los productos y precios, etc, los cuales son, a su vez, analizados para generar informes de alto valor sobre el comportamiento de los clientes.

Todas las prácticas de este año están orientadas a este problema, analizar el registro de actividad de los clientes de una web de venta de productos, y elaborar informes de comportamiento de las ventas.

Figura 1: Arquitectura de las prácticas de MP 2021

Para ello, se va a desarrolar una arquitectura de clases (ver Figura 1) que se irán desarrollando progresivamente a lo largo de la asignatura, desde las más primitivas hasta las más abstractas.

1.1. Los datos

Para la realización de las prácticas trabajaremos sobre un conjunto de datos reales que contiene los registros de actividad, durante un periodo

Registro de actividad

Date Time	Event Type	Product ID	Category ID	Category Code	Brand	Price	User ID	Session ID
string	string	string	string	string	string	double	string	string

Figura 2: Estructura del registro de actividad

definido de una tienda de cosméticos, llegando a trabajar con más de 75,000 registros.

Los datos han sido descargados desde la plataforma Kaggle[1], y en ella se pueden encontrar más detalles y diferentes periodos de tiempo.

Varias son las características de este dataset. En primer lugar, muchos de los datos están incompletos y, en segundo lugar, el tamaño medido en número de registros, como ya comprobaremos, nos plantea algunos retos de **eficiencia** tanto en el procesamiento de los datos como, en la gestión de memoria para los datos. Lo que podría ser aceptable para unos datos de juguete, no es soportable para unos datos reales. ¡Bienvenido al mundo real!

Cada fila de un dataset representa un evento, y un evento contiene la siguiente información:

1. DateTime. Una fecha que contiene el instante de la acción, formado por fecha y hora en el formato UTC. Un ejemplo de datetime,

$$2019 - 10 - 01 \;\; 00 : 15 : 06 \; UTC$$

2. Event type. Cada evento puede ser uno entre cuatro tipos de acciones: “view”,“cart”,“purchase”,“remove_from_cart”.
3. Product_id. El identificador del producto.
4. Category_id. El identificador de la categoría a la que pertenece el producto.
5. Category_code. Un descriptor de la categoría.
6. Brand. Marca del producto.
7. Price. Precio del producto.
8. User_id. Identificador de un usuario.
9. Session_id. Identificador de sesión (cada conexión al servidor web es una sesión y todas las operaciones que lleve a cabo en la misma sesión compartirán el mismo código de usuario y número de sesión).

Todos los conjuntos de datos que vamos a utilizar están en formato csv, esto es, se trata de ficheros de texto, donde cada evento es un línea y los campos están separados por (,). Estos ficheros se encuentran en la carpeta

```
./tests/validation
```

[1] Kaggle (Abrir en navegador →)

Date Time	Event Type	Product ID	Category ID	Category Code	Brand	Price	User ID	Session ID
string	string	string	string	string	string	double	string	string
2019-10-01 00:15:06 UTC	cart	5869134	17839990		cosmoprofi	6.35	554342223	0b974342-1a53-41c1

```
2019-10-01 00:15:06 UTC , cart , 5869134 , 17839990 ,   , cosmoprofi ,  6.35 , 554342223 , 0b974342-1a53-41c1
```

Figura 3: Un ejemplo de evento del dataset y su composición en el fichero de datos

Los registros están ordenados por la fecha en la que se produjeron, por lo que los registros de un determinado usuario en la misma sesión no tienen por qué encontrarse de forma consecutiva, sino que se van registrando conforme los usuarios conectados van realizando acciones.

No todos los datos están presentes en un evento, en el dataset. Los campos indispensables son DateTime, Product_id, User_id, y Session_id. Como podemos observar en el ejemplo anterior, no hay valor para Category_code, es decir, no está presente.

1.2. Arquitectura de la práctica

La práctica `Shopping` se ha diseñado como una arquitectura por capas, en la que las capas más internas de la misma representan las estructuras más sencillas, sobre las cuales se asientan las capas más externas, con las estructuras más complejas. La Figura 1 muestra el diseño de la arquitectura que se va a emplear y que se va a ir desarrollando progresivamente en esta y las siguientes sesiones de prácticas. En total, estas son las clases que nos vamos a encontrar y que tendremos que implementar.

A DateTime.cpp
: Implementa la clase DateTime, compuesta por año, mes, día, hora, minutos y segundos. Toda instancia de esta clase ha de ser correcta, esto es, una fecha ha de ser válida y el tiempo dentro de los rangos correctos.

B Event.cpp
: Implementa la clase Event, que contiene la información de cada acción registrada en el sitio web.

C EventSet.cpp
: Implementa la estructura para almacenar un conjunto de eventos. Como comprobaremos muchos menos que los deseables. Inicialmente usaremos arrays estáticos.

D Pair.cpp
: Estructura para almacenar una clave de búsqueda y una posición en el EventSet.

E Index.cpp
: Array de claves, que se va a utilizar como índice para la búsqueda y recuperación eficiente en el EventSet.

C, E EventSet-Opt.cpp
Manteniendo la interfaz anterior, se cambia la implementación para alojarlas de forma más eficiente en memoria dinámica.

F Report.cpp
Implementa la estructura para almacenar una matriz que nos va a permitir realizar estadísticas y comparativas.

Este trabajo progresivo se ha planificado en hitos sucesivos con entregas en Prado.

2. Shopping1, primer nivel de la práctica a entregar

2.1. Descripción de la práctica

Se deberá crear un proyecto Netbeans compuesto por los siguientes ficheros fuente incompletos.

- **DateTime.h**
  Revisar y completar las declaraciones de los métodos, respetando el número y tipo de los parámetros, pero estableciendo una adecuada comunicación entre módulos, esto es: métodos const o no const, parámetros const o no const, parámetros por valor o por referencia etc. de acuerdo con las especificaciones dadas en la cabecera de cada método, para que pueda funcionar correctamente.

- **DateTime.cpp**
  Completar la definición de todos los métodos de la clase, de acuerdo con las especificaciones dadas en el fichero de cabeceras.

- **Event.h**
  Revisar y completar las declaraciones de los métodos, y funciones externas según criterios de comunicación entre módulos: (métodos const/ no const, parámetros const/ no const, parámetros por valor/ referencia y tipo) de acuerdo con las especificaciones dadas en la cabecera de cada módulo (método o función).

- **Event.cpp**
  Implementar todos los métodos y funciones que faltan. Prestar atención al funcionamiento de los métodos y funciones descritos en los comentarios.

- **main.cpp**
  Aunque las lecturas que se realicen se hacen desde el teclado, las entradas al ser tan extensas, se tomarán desde fichero, **con redireccionamiento de la entrada < ; ya utilizado en las prácticas anteriores**. Completar el código para realizar el programa que se describe en los comentarios:

```
5
2019-10-01 00:15:06 UTC,cart,5869134,1783999064136745198,,cosmoprofi,6.35,554342223,0b974342-1a53-41c1-a426-23130e770f4b
2019-10-01 00:17:10 UTC,cart,5787018,1487580006644188066,,,6.33,554342223,0b974342-1a53-41c1-a426-23130e770f4b
2019-10-01 00:21:02 UTC,cart,5836843,1487580009261432856,,pnb,0.71,554342223,0b974342-1a53-41c1-a426-23130e770f4b
2019-10-01 00:22:24 UTC,cart,5755171,1487580009387261981,,,2.48,554342223,0b974342-1a53-41c1-a426-23130e770f4b
2019-10-01 00:22:56 UTC,cart,5691026,1487580009387261981,,,2.86,554342223,0b974342-1a53-41c1-a426-23130e770f4b
```

Figura 4: Contenido del fichero de datos `./tests/validation/Ecommerce5.keyboard`

```
./dist/Debug/GNU-Linux/shopping1 < tests/validation/ECommerce5.keyboard

Number of events to read
2019-10-01 00:15:06 UTC,cart,5869134,1783999064136745198,,cosmoprofi,6.350000,554342223,0b974342-1a53-41c1
2019-10-01 00:17:10 UTC,cart,5787018,1487580006644188066,,,6.330000,554342223,0b974342-1a53-41c1
2019-10-01 00:21:02 UTC,cart,5836843,1487580009261432856,,pnb,0.710000,554342223,0b974342-1a53-41c1
2019-10-01 00:22:24 UTC,cart,5755171,1487580009387261981,,,2.480000,554342223,0b974342-1a53-41c1-
2019-10-01 00:22:56 UTC,cart,5691026,1487580009387261981,,,2.860000,554342223,0b974342-1a53-41c1
Records read: 5

Activity found: SUNDAY(0) MONDAY(0) TUESDAY(5) WEDNESDAY(0) THURSDAY(0) FRIDAY(0) SATURDAY(0)
Valid records: 5
Max activity: 5
Day of Max activity: TUESDAY
```

Figura 5: Una ejecución de ejemplo que muestra la llamada al programa (en azul) que lee los datos de `/tests/validation/ECommerce5.keyboard` y la salida esperada. Solo las líneas rojas deben de mostrarse en `CVAL`

1. Utilizar un array estático de Eventos para almacenar una secuencia de eventos leídos desde el teclado, redirigido desde un archivo de datos en la carpeta `./tests/validation/*.keyboard`.
2. Leer el número máximo de registros válidos para ser almacenados en el array.
3. Si el evento leído es correcto (utiliza el método isEmpty() para detectar registros no válidos), se almacena en el array, en caso contrario, se descarta. Solo cuentan los registros válidos.
4. A continuación, calcula el número de eventos, de todo tipo, encontrados para cada día de la semana (implementar la función local computeActivity() que también muestra el resultado en la pantalla).
5. A continuación, encuentra cuál es el día de mayor actividad registrada y lo muestra en pantalla.

2.2. Un caso de ejemplo

La figura 5 muestra un ejemplo de ejecución, con la llamada al programa en azul, en el que las salidas más importantes, en color rojo, no se mostrarán por `cout` sino por `CVAL`, un flujo de salida, como `cout` pero que se usará durante la fase de tests para comprobar que la salida de datos es la correcta. Es decir, los programas podrán realizar todos los `cout` que estimen oportunos, pero para comprobar que la ejecución es correcta, se comparará la salida obtenida por `CVAL` con respecto a la esperada.

2.3. Tests completos de la práctica

Se recomienda repasar primero los videotutoriales completos sobre este la metodología TDD en Google Drive

(Abrir en navegador →)

y la preparación del entorno de trabajo con MPTest

(Abrir en navegador →)

Esta primera parte de la práctica se ha diseñado con tres niveles de testeo, que incluyen tanto tests unitarios como de integración (Ver Anexo II y Sección 3).

- Nivel Básico: 10 tests
- Nivel Intermedio: 2 tests
- Nivel Avanzado: 3 tests

Y este debería ser el resultado del test de la aplicación satisfaciendo todos los tests que se han diseñado.

make test

```
[==========] Running 15 tests from 3 test suites.
[----------] Global test environment set-up.
[----------] 10 tests from _01_Basics
[ RUN      ] _01_Basics.DateTime_Constructors
[       OK ] _01_Basics.DateTime_Constructors (1 ms)
[ RUN      ] _01_Basics.DateTime_getters
[       OK ] _01_Basics.DateTime_getters (0 ms)
[ RUN      ] _01_Basics.DateTime_set
[       OK ] _01_Basics.DateTime_set (0 ms)
[ RUN      ] _01_Basics.Event_ConstructorBase
[       OK ] _01_Basics.Event_ConstructorBase (0 ms)
[ RUN      ] _01_Basics.Event_Setters_getters
[       OK ] _01_Basics.Event_Setters_getters (1 ms)
[ RUN      ] _01_Basics.Integration_ECommerce5
[       OK ] _01_Basics.Integration_ECommerce5 (10 ms)
[ RUN      ] _01_Basics.Integration_EMPTY
[       OK ] _01_Basics.Integration_EMPTY (8 ms)
[ RUN      ] _01_Basics.Integration_ECommerce49
[       OK ] _01_Basics.Integration_ECommerce49 (6 ms)
[ RUN      ] _01_Basics.Integration_ECommerce_2019_Q4_200
[       OK ] _01_Basics.Integration_ECommerce_2019_Q4_200 (13 ms)
[ RUN      ] _01_Basics.Integration_ECommerce_all_all_200
[       OK ] _01_Basics.Integration_ECommerce_all_all_200 (8 ms)
[----------] 10 tests from _01_Basics (49 ms total)

[----------] 2 tests from _02_Intermediate
[ RUN      ] _02_Intermediate.DateTime_isBefore
[       OK ] _02_Intermediate.DateTime_isBefore (1 ms)
[ RUN      ] _02_Intermediate.DateTime_weekDay
[       OK ] _02_Intermediate.DateTime_weekDay (0 ms)
[----------] 2 tests from _02_Intermediate (1 ms total)

[----------] 3 tests from _03_Advanced
[ RUN      ] _03_Advanced.DateTime_BadValues
[       OK ] _03_Advanced.DateTime_BadValues (1 ms)
[ RUN      ] _03_Advanced.Event_setType_Bad_Values
[       OK ] _03_Advanced.Event_setType_Bad_Values (0 ms)
[ RUN      ] _03_Advanced.Event_Others_Bad_Values
[       OK ] _03_Advanced.Event_Others_Bad_Values (1 ms)
[----------] 3 tests from _03_Advanced (2 ms total)

[----------] Global test environment tear-down
[==========] 15 tests from 3 test suites ran. (52 ms total)
[  PASSED  ] 15 tests.
```

3. TESTS DOCUMENTATION FOR PROJECT Shopping1

3.1. _01_Basics

3.1.1. DateTime_Constructors

1. A newly create instance of DateTime gives the default date
2. A newly created instance of DateTime by using a string gives the same string

3.1.2. DateTime_getters

1. The year of the default datetime must be 2021
2. The month of the default datetime must be 1
3. The day of the default datetime must be 1
4. The hour of the default datetime must be 0
5. The minutes of the default datetime must be 0
6. The seconds of the default datetime must be 0

3.1.3. DateTime_set

1. Setting an instance of DateTime with a valid string gives the same date";

3.1.4. Event_ConstructorBase

1. A newly created instance of Event gives the default event
2. A newly created instance of Event is consdered empty
3. A newly create instance of Event, initializaed with a string. gives the same string
4. Setting the productid of an event to a good value X gives X

3.1.5. Event_Setters_getters

1. Setting the datetime of an event to TODAY gives TODAY
2. Setting the type of an event to a good value X gives X
3. Setting the productid of an event to a good value X gives X
4. Setting the categoryID of an event to a good value X gives X
5. Setting the category code of an event to a good value X gives X
6. Setting the brand of an event to a good value X gives X
7. Setting the price of an event to a good value X gives X

3.1.6. Integration_ECommerce5

1. Simple test with five (5) records only

3.1.7. Integration_EMPTY

1. Simple tet with an empty set of records

3.1.8. Integration_ECommerce49

1. Test with 49 records

3.1.9. Integration_ECommerce_2019_Q4_200

1. Test with 200 records of 2019-Q4

3.1.10. Integration_ECommerce_all_all_200

1. Test with 200 records amongst all available years

3.2. _02_Intermediate

3.2.1. DateTime_isBefore

1. A DateTime cannot be before itself
2. Default date is before today
3. Today is not before the Default date

3.2.2. DateTime_weekDay

1. Today must be Thursday
2. Tomorrow is Friday
3. Yesterday was Wednesday

3.3. _03_Advanced

3.3.1. DateTime_BadValues

1. Setting a bad date or time gives the default datetime
2. Setting a bad date or time gives the default datetime
3. Setting a bad date or time gives the default datetime
4. Setting a bad date or time gives the default datetime
5. Setting a bad date or time gives the default datetime
6. Setting a bad date or time gives the default datetime
7. Setting a bad date or time gives the default datetime
8. Setting a bad date or time gives the default datetime
9. Setting a bad date or time gives the default datetime
10. Setting a date with the incorrect format throws an exception

3.3.2. Event_setType_Bad_Values

1. Setting the type of an event to a good value X gives X
2. Setting the type of an event to a good value X gives X

3.3.3. Event_Others_Bad_Values

1. Setting the productid of an event to a good value X gives X
2. Setting the productid of an event to a good value X gives X
3. Setting the productid of an event to a good value X gives X

Anexo I. Breve repaso de la clase string en C++

La clase `string` nos permite manipular cadenas de caracteres de forma simple y segura. En esta sección repasaremos brevemente algunos de los métodos de la clase (para mas información podéis consultar la referencia de la clase en http://www.cplusplus.com/reference/string/string/), en especial los métodos `find` y `substr` serán de utilidad para extraer los campos de un evento.

Para poder declarar un objeto `string` lo primero que tendremos que hacer es incluir el fichero cabecera, pudiendo asignarle valores en el constructor o leerlos desde la entrada con `cin`:

```
#include <string>
using namespace std;
string s1;
string s2("Valor inicial de la cadena");

cin >> s1; // lee hasta encontrar el separador
```

Si necesitamos leer una línea entera (hasta el carácter nueva línea `'\n'`) podemos hacerlo utilizando la función `getline` que recibe como entrada un flujo de entrada (como puede ser la entrada estándar, con `cin`)

```
getline(cin,s1);
```

- Concatenar:

  Los string se pueden concatenar utilizando el operador suma (+)

```
string s1 = "esto es ";
string s2 = "un ejemplo de suma";
string s3 = s1+s2;
cout << s3 << endl; // "esto es un ejemplo de suma"
s3 += " de cadenas";
cout << s3; // "esto es un ejemplo de suma de cadenas"
```

- Longitud de un string:

  Hay dos métodos para consultar la longitud de un string, `size` y `length`

```
cout << s3.size();
cout << s3.length();
```

- Acceso a los caracteres individuales:

  Los caracteres de un string se indexan igual que los elementos de un array de caracteres, las posiciones válidas del mismo van desde 0 hasta `size() - 1`

```
for (int i=0; i<s3.size();i++)
   cout << s3[i] << endl;     // "esto es un ejemplo de suma de cadenas"
for (int i=0; i<s3.size();i++)
   s3[i] = s3[i]+1;  // asignamos siguiente caracter
cout << s3;          // "ftup!ft!vo!fkfnqmp!ef!tvnb!ef!dbefobtok"
```

- Comparación entre string:

  La comparación entre strings se hace utilizando los operadores relacionales `==,!=,<,>,<=,>=`

```
string passwd;

getline(cin, passwd, '\n');
if(passwd != "aabbcc")
{
    cout<<"Acceso denegado";
}
```

- Búsqueda:

  La clase `string` nos proporciona distintos mecanismos de búsqueda en el string entre los que destacan el método `find()` (consultar la referencia para ver otras alternativas que pueden ser útiles).

  El método

  `int string::find(const string & s,unsigned int  pos)`

  toma como entrada una cadena `s` y una posición `pos` y empieza a buscar la primera ocurrencia de la cadena `s` a partir de la posición `pos`, devolviendo la posición de inicio de la primera ocurrencia o bien un valor especial `string::npos` si no la encuentra

```
string cad;
int i = 0;
int cuantas = 0;

cad = "esto es un ejemplo especifico de find en cadenas";

for(i= cad.find("es",0); i != string::npos; i= cad.find("es",i))
{
    cuantas++;
    i++;  // Avanzamos i para buscar a partir de la siguiente posición
}
cout<<cuantas;
```

  Nota: el método lo podremos utilizar en nuestra práctica para buscar el separador `","` en una línea del fichero CSV.

- Subcadenas:

  El método `substr(int pos, int tama)` nos permite crea una cadena que contiene la porción de string que empieza en la posición `pos` con tamaño `tama`. Por ejemplo, para extraer los 10 primeros caracteres de un string podemos utilizar

```
string s = "abcdefghijklmnop";
string primeros10  = s.substr(0, 10);
```

  Nota: el método lo podremos utilizar para extraer los distintos campos del fichero CSV, esto es, los elementos entre dos separadores (comas consecutivas).

- Transformar cadena a números (y viceversa):

  Para realizar estas transformaciones, la biblioteca string (a partir de estándar C++11) proporciona una serie de funciones (no métodos de la clase), permitiendo convertir la cadena a entero, flotante, double, etc. (mirar la documentación de la clase string) y viceversa. El siguiente ejemplo nos muestra cómo trabajar con doubles:

```
string valor ("365.24");
double x = stod (valor);
double dx = x*2;
cout << "El doble de "<< x <<" es " << dx << endl;

string doblevalor = to_string(dx);
cout << "La cadena representando al doble es "<< doblevalor;
```

Nota: Lo utilizaremos para transformar el campo precio del CSV a valor numérico.

Anexo II. Introducción a la metodología Test Driven Development (TDD)

TDD es una metodología de desarrollo de aplicaciones que se basa en escribir, en primer lugar, los tests que debe de pasar correctamente el software que se va a crear y, posteriormente, crear ese software con el objetivo fundamental de que pase todos los tests para, posteriormente, adaptarlo a un diseño y/o arquitectura específica. Eso sí, sin dejar de pasar nunca todos los tests. Quizás los dos objetivos fundamentales de TDD sean buscar el diseño más sencillo posible, evitando toda complejidad innecesaria (KISS princple[2]) y, por otro lado, proporcionar confianza en el desarrollo de software.

Hay varios tipos de tests, que varían según la aplicación que se esté desarrollando, entre los que podemos distinguir los más básicos:

- Tests Unitarios[3]. Cuyo objetivo es comprobar que cada unidad funcional mínima que se construya, ya sea una función o un método, cumple una serie de buenas propiedades. Así, podemos considerar que la implementación de una clase es correcta si todos sus métodos han sido probados individualmente, lo cual ofrece una base de trabajo garantizada desde la que desarrollar nuevas clases o construir el programa completo.
- Tests de Integración[4]. Cuyo objetivo es comprobar que la integración de todas las clases construidas, la entrada y salida de datos, sigue siendo correcta, partiendo de la base de que las clases se han probado correctas con tests unitarios.

[2]KISS = Keep It Simple Stupid, (Abrir en navegador →)
[3]Unit Testing (Abrir en navegador →)
[4]Integration Testing (Abrir en navegador →)

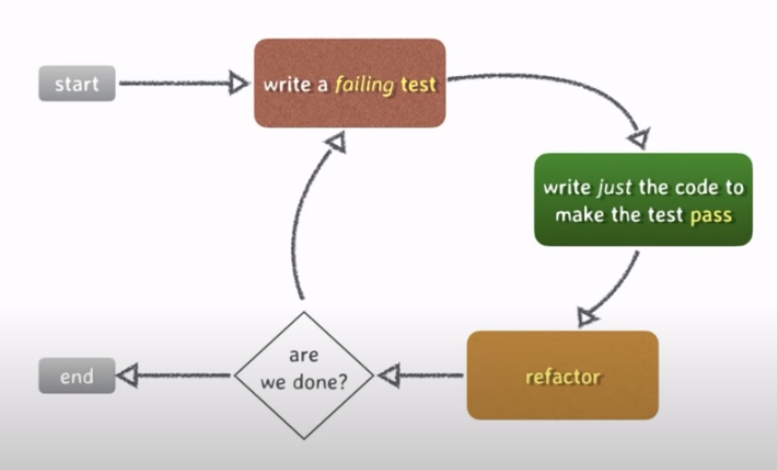

Figura 6: Ciclo de vida de TDD. Phil Nash, CppCon The C++ Conference, 2020. (Abrir en navegador →)

Los pasos a seguir en la metodología TDD, que no son muy diferentes del proceso tradicional, solo que están más ordenados (ver Figura 6), son los siguientes (en negrilla los que debe de hacer el estudiante, en tipografía normal los pasos que aporta el profesor).

1. Escribir el diseño de la API de la clase, solo los métodos.
2. Escribir los tests unitaros y de integración.
3. **Escribir el código de los métodos hasta que pasen todos los tests unitarios, sea como sea.**
4. **Cuando pasen todos los tests, intentar refactorizar el código para mejorarlo, en cuyo caso habría que volver al paso 3 para testear los cambios hechos.**
5. **Escribir el main() para que pase los tests de integración, sea como sea.**
6. **Es posible que haya que refactorizar alguna clase previa. Si es así, volver al paso 3.**
7. **Si ha pasado todos los tests de integración, intentar refactorizar el código para mejorarlo y volver al paso 5.**
8. **Compilar la versión definitiva.**

3.4. Configuración de las prácticas

3.4.1. El espacio de trabajo

MPTest.zip : Este es el material inicialmente entregado a los estudiantes en Prado, organizado en carpetas. Contamos con que lo descomprimas en **ProyectosNetBeans :** Carpeta raíz en la que se van a crear todos los proyectos de la asignatura.

```
MPTest.zip
├── googletest-master
│   └── googletest
│       └── include
├── Gtest
├── MPTest
│   └── include
└── Scripts
```

googletest-master: Rama principal de desarrollo de GoogleTest en Git (no es necesario que sea la última versión). **GTest:** Proyecto de NetBeans que permite generar, a cada estudiante, una versión enlazable de GoogleTest para su máquina y sistema operativo. **MPTest:** Extensión para cubrir los tests de integración en MP. Por ahora esta extensión es un único fichero de cabeceras el cual contiene las macros necesarias para ello. **Scripts:** Una serie de scripts Bash de apoyo a las funciones de NetBeans. Se recomienda empezar por crear la carpeta ProyectosNetBeans y descomprimir el fichero de material previo **MPTest.zip** (Abrir en navegador →) dentro de ella, como muestra la figura a la izquierda. Abrir el proyecto **Gtest** y compilarlo (Clean & Bluild).

Para tener bien ordenado el contenido de cada proyecto Shopping, se va a dividir en las siguientes carpetas:

```
Shopping
├── build
├── dist
├── data
├── doc
│   └── doc.doxy
├── include
├── Makefile
├── nbproject
├── scripts
├── src
├── tests
└── zip
```

Figura 7: Estructura interna de cada proyecto Shopping

build Es una carpeta temporal que contiene código precompilado y pendiente de enlazar

dist Contiene los binarios ejecutables. Vamos a generar dos versiones por cada programa:

- **Release** Es la versión más eficiente y que menos espacio ocupa. Es la que debería entregarse al cliente.
- **Debug** Es la versión sobre la que se depuran errores y se corrigen defectos. Es más grande porque el binario contiene información extra para poder usar el depurador. Puede llegar a ser el doble de grande que la anterior.

data Ficheros de datos, bases de datos, ficheros de configuración que pueda necesitar el programa durante su ejecución, que no sean para testearla.

doc Aquí se almacena la documentación del proyecto, tanto del código, como la generada por los tests.

include Contiene los ficheros de cabeceras **.h**.

nbproject Es la carpeta que utiliza NetBeans para almacenar la configuración del proyecto

scripts En esta carpeta colocaremos los scripts de apoyo a Netbeans que se han desarrollado en esta asignatura y se puede ampliar con otros más.

src Contiene los principales ficheros fuente del proyecto **.cpp**, sin incluir los ficheros de test, que también son **.cpp** , que van en la siguiente carpeta

tests Carpeta dedicada a los tests unitarios y de integración. Dentro de ella se encuentran, en el primer nivel, los ficheros fuente de los tests **.cpp**, la carpeta `output` que se utiliza como carpeta temporal durante los tests y la carpeta `validation` que se utiliza para **ficheros de datos únicamente usados en la validación de la aplicación** completamente integrada.

zip Carpeta para copias de seguridad del proyecto.

3.5. Configurar el proyecto en NetBeans

Se recomienda repasar primero los videotutoriales completos sobre este tema en Google Drive (Abrir en navegador →)

1. Crear el proyecto en NetBeans como `C++ Application`.
   a) Eliminar el entorno de trabajo `Release` y dejar sólo `Debug`. Aunque en adelante se crearán otros entornos de trabajo, en lo siguiente se trabajará en `Debug`.
2. Copiar la carpeta `./scripts`, copiar el script `runUpdate.sh` dentro y ejecutarlo[5] Esto crea la estructura de carpetas descrita anteriormente y actualiza los scripts que pudiese haber nuevos. Estos cambios se han llevado a cabo através del SO, compruébalo con la instrucción unix: `ls`, sin embargo no aparecen en Netbeans... Para que se refleje en el IDE, desplegar opción de menú Source `->Scan for external changes` y ya aparecen las nuevas carpetas...
3. Colocar cada fichero en su sitio, según indica la imagen anterior.
4. Propiedades de proyecto. Compilador
   a) Poner el estándar a C++14
   b) Añadir los `Include Directories ...` siguientes:
   del propio proyecto **./include**
   las de testeo **../MPTest/include**,
   ../googletest-master/googletest/include
   y el resto de librerías externas que pueda utilizar el proyecto.
5. Desde la vista lógica del proyecto.

[5]Comprueba que tenga permiso de ejecución, se cambia con la instrucción unix: `chmod +x` , en la terminal.

a) En `Header Files`, `Add existing item` añadir los ficheros `DateTime.h` y `Event.h`.

b) En `Source Files`, `Add existing item` añadir los ficheros `main.cpp` y `Event.cpp`.

6. Sobre el nombre del proyecto, botón derecho `Set As Main Project` o desde el menú principal `Run` – `Set Main Project` y seleccionar este proyecto.

7. Con esto se puede ejecutar el proyecto normalmente. Obviamente, no hace nada porque está vacío y no se llama a ninguna función. A partir de aquí empezaría el desarrollo del proyecto, el cual cada programador seguirá un orden determinado y, probablemente, diferente a todos los demás, hasta que la aplicación esté terminada. En ese momento habrá que probarla, para ver que todo funciona como se espera, en un proceso que, de nuevo, dependerá de cada programador en concreto.

La práctica deberá ser entregada en Prado, en la fecha que se indica en cada entrega, y consistirá en un fichero ZIP del proyecto (para ello se sugiere utilizar el script `runZipProject.sh`).