



Guion de prácticas

Shopping3

Pair, Index

Abril de 2021



Metodología de la Programación

DGIM-GII-GADE

Curso 2020/2021

Índice

1. Descripción	6
1.1. Los datos	6
1.2. Arquitectura de la práctica (recordatorio)	6
2. Shopping3, práctica a entregar	8
2.1. Descripción de la práctica	10
2.2. Un caso de ejemplo	13
2.3. Tests completos de la práctica	16
2.4. Configuración de la práctica	18
2.5. Entrega de la práctica	19
3. TESTS DOCUMENTATION FOR PROJECT Shopping3	20
3.1. _01_Basics	20
3.1.1. DateTime_Constructors	20
3.1.2. DateTime_getters	20
3.1.3. DateTime_set	20
3.1.4. DateTime_sameDay	20
3.1.5. Event_ConstructorBase	20
3.1.6. Event_Setters_getters	20
3.1.7. EventSet_Constructor	21
3.1.8. EventSet_add_event	21
3.1.9. EventSet_add_line	21
3.1.10. EventSet_at_basic	21
3.1.11. Pair_Constructors	21
3.1.12. Pair_isEmpty	21
3.1.13. Pair_setters	21
3.1.14. Pair_getters	22
3.1.15. Index_Constructors	22
3.1.16. Index_get!OnWhich	22
3.1.17. Index_clear	22
3.1.18. Integrated_5_records	22
3.1.19. Integrated_30_records	22
3.1.20. Integrated_41_records	22
3.1.21. Integrated_162_records	22
3.1.22. Integrated_926_records	22
3.1.23. Integrated_Args_5_records	23
3.1.24. Integrated_Args_30_records	23
3.1.25. Integrated_Args_41_records	23
3.1.26. Integrated_Args_162_records	23
3.1.27. Integrated_Args_926_records	23
3.2. _02_Intermediate	23
3.2.1. DateTime_isBefore	23
3.2.2. DateTime_weekDay	23
3.2.3. Event_getField	23
3.2.4. EventSet_add_event_partial	24
3.2.5. EventSet_at_intermediate	24
3.2.6. Index_10x10_just_build	24



3.2.7. Index_B_BxU_build_at	24
3.2.8. Index_U_BxU_build_at	24
3.2.9. Integrated_EMPTY	24
3.2.10. Integrated_ErrorLoading	24
3.3. _03_Advanced	24
3.3.1. DateTime_BadValues	24
3.3.2. Event_setType_Bad_Values	25
3.3.3. Event_Others_Bad_Values	25
3.3.4. EventSet_add_event_full	25
3.3.5. EventSet_at_advanced	25
3.3.6. EventSet_externalfunctions	25
3.3.7. EventSet_write	25
3.3.8. EventSet_read	26
3.3.9. Index_U_BxU_bounds	26
3.3.10. Index_B_BxU_bounds	26
3.3.11. Index_add	26
3.3.12. Index_B_BxU_rawFilterIndex	26
3.3.13. Index_U_BxU_rawFilterIndex	26
3.3.14. Index_Type_BxU_rawFilterIndex	26
3.3.15. Index_DateTime_BxU_rawFilterIndex	26
3.3.16. Index_BxU_sumPrice	27
3.3.17. Integrated_ErrorData	27
3.3.18. Integrated_ErrorSaving	27
3.3.19. Integrated_Args_no_open	27
3.3.20. Integrated_Args_error_data	27
3.3.21. Integrated_Args_missing_arg	27
3.3.22. Integrated_Args_error_bad_arg	27
A. Argumentos de main	28
B. La biblioteca fstream	30
B.1. Primeros Pasos	30
B.2. Abriendo y cerrando ficheros	31
B.3. Ejemplo de uso de archivos	32
C. Ejemplo de uso de las clases Pair e Index	33

1. Descripción

Recordemos que todas las prácticas de este año están orientadas al problema de ventas por internet, analizando el registro de actividad de los clientes de una web de venta de productos, y elaborando informes de comportamiento de las ventas.

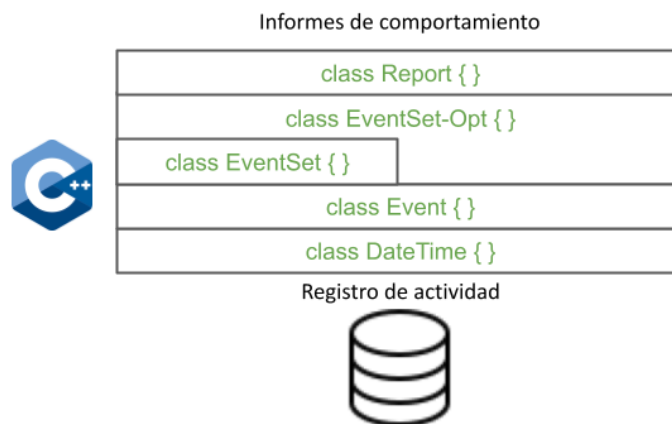


Figura 1: Arquitectura de las prácticas de MP 2021

Para ello, se va a implementar 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.

En las prácticas anteriores (Shopping1 y shopping2) ya se implementaron las clases `DateTime`, `Event` y `EventSet`, así como un programa principal para integrar y usar dichas clases en varios conjuntos de datos de prueba. En esta práctica, se van a desarrollar las clases `Pair` e `Index`, que nos permitirán hacer búsquedas más eficientes sobre nuestros conjuntos de datos. Además, se van a visualizar los resultados con la biblioteca `DataTable`, la cual se os proporciona junto con la práctica.

1.1. Los datos

En esta práctica seguiremos trabajando con un pequeño fragmento del conjunto de datos reales (alrededor de 75,000 registros) descargados desde la plataforma Kaggle¹. En la Figura 2 se encuentra un recordatorio del formato de dichos datos en el dataset. Recuerda que un evento puede tener algunos valores vacíos. Los únicos datos que siempre estarán presentes serán `DateTime`, `Product ID`, `UserID`, y `SessionID`.

1.2. Arquitectura de la práctica (recordatorio)

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,

¹Kaggle ([Abrir en navegador →](#))



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

```

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 3: Contenido del fichero de datos
./tests/validation/Ecommerce5.keyboard

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. Durante el desarrollo de la asignatura se implementarán un total de 7 clases que dotarán al programa de la funcionalidad deseada. Concretamente las clases que nos vamos a encontrar y que tendremos que implementar son:

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-Op.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. Shopping3, práctica a entregar

Tras el desarrollo de las prácticas anteriores (Shopping1 y Shopping2) tenemos las tres primeras capas (ver Figura 1) de nuestra arquitectura desarrollada. El programa desarrollado hasta el momento nos permite hacer consultas sencillas sobre nuestro conjunto de datos. Así en la Shopping2 pudimos filtrar nuestro conjunto de datos por tipo de evento para consultar únicamente los eventos asociados a una compra, y obtener el precio total para cada una de las marcas que habían realizado alguna venta. Sin embargo, conforme nuestro conjunto de datos aumenta de tamaño, trabajar con él se vuelve más lento y agota más recursos. Por ello, a la hora de realizar informes avanzados o un análisis más elaborado de los datos, es interesante disponer de estructuras adicionales que nos permitan trabajar con nuestros datos de una forma más eficiente.

Índices. ¿Qué son?

Como habéis podido comprobar, en los archivos de ejemplo con los que hemos trabajado hasta el momento, nuestros eventos están ordenados por fecha; cada vez que se produce un evento nuevo, en el servidor se añade la información de dicho evento al final del archivo de registros.

No obstante, cuando queremos extraer estadísticas de ventas (*purchase*), consultas (*view*) etc., normalmente no estamos interesados en la información temporal (organizada por `DateTime`) si no que, pasan a ser más relevantes otros campos de nuestro histórico de eventos. En particular, en nuestra aplicación tenemos un gran interés en generar informes agrupados por usuarios (`UserID`) o agrupados por marcas (`Brand`) que nos permitan extraer conclusiones y poder hacer promociones de marcas y recomendaciones a usuarios

Por otro lado, muchas consultas solo hacen referencia a una pequeña parte de los registros de un archivo. Por ejemplo, eventos relativos a un determinado usuario o relativos a una marca en particular. No es eficiente que tengamos que leer todos los eventos para encontrar los eventos del `userID = 446419295` o los eventos asociados a `Brand = runail`. Lo más adecuado sería que pudiéramos localizar directamente estos registros. Para facilitar esta forma de acceso vamos a usar índices.

Un índice, para nuestro conjunto de eventos, es una estructura de datos que funciona como el índice de cualquier libro. Si se quiere buscar información concreta, se busca en el índice ordenado alfabéticamente, lo que facilita la búsqueda, y dicho índice nos indica la(s) página(s) donde encontrar la información buscada. Además, el índice es mucho más pequeño que el libro, con lo que el esfuerzo de búsqueda es aún menor.

El índice que vamos a diseñar juega el mismo papel que el índice de un libro. Por ejemplo, vamos a almacenar una lista de marcas, ordenadas en orden alfabético, junto con cada una de las entradas en el conjunto de eventos donde está implicada cada marca. Así, cuando se requiere una marca determinada, se busca de forma eficiente en el índice ² para localizar las entradas en el conjunto de eventos y poder recuperar rápidamente los registros de dicha marca.

De esta forma, un índice está formado por un conjunto de registros, que va a ser consultado por un campo, la clave de búsqueda, *key*, el string por el que está ordenado y nos va a devolver unas posiciones *Pos*, uno por valor de clave, dónde recuperar los registros de datos en el conjunto de eventos. Inicialmente, vamos a considerar un registro en el índice por cada evento en el conjunto de eventos, esto es, si n es el tamaño del conjunto de eventos, n es el número de registros en el índice ³. Cada registro (le llamaremos *Pair*) del índice, está compuesto por un campo string, *key* y un entero *pos* la posición dentro del conjunto de eventos, un valor entre 0 y $n - 1$.

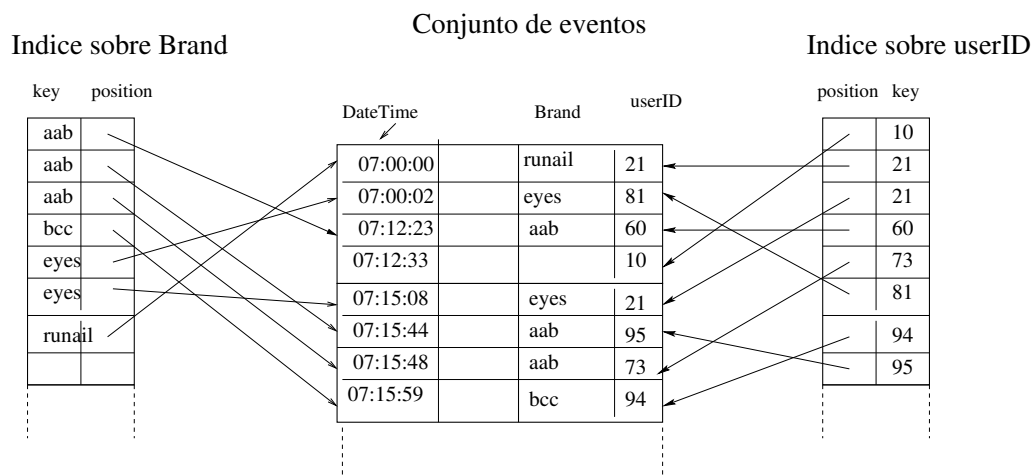


Figura 4: Ejemplo de índices sobre userID y sobre Brand apartir de un conjunto de Eventos.

Todo lo comentado sobre marcas se puede aplicar igualmente sobre la cadena `userID`, pudiendo montarse uno o varios índices sobre nuestro conjunto de eventos como puede verse en la figura 4. Un índice como los que vamos a utilizar en esta práctica, aún ordenado por clave, puede tener valores duplicados. Es decir, pueden existir varios

²Se puede utilizar la búsqueda binaria sobre las cadenas de marca (Brand), ya que existe un orden entre las cadenas, ejemplo: $Abc < Abd$.

³Esto solo es cierto para `userID`, ya que este campo es obligatorio, pero en el caso de Brand, algunos eventos tienen valor desconocido y no se almacena ese valor como *key* en el índice. Esa es la razón por la que la longitud de un índice sobre Brand puede ser menor que n .

Pair con el mismo valor de clave, como se puede comprobar en la mencionada figura. Existirá un valor por cada evento del mismo usuario, o de la misma marca, en instantes diferentes. Cuando se crea el índice, se mantiene el orden por valor de clave, pero a igualdad de clave el nuevo *Pair* se inserta antes de un valor de clave superior.⁴ Para saber cómo se monta un índice sobre un conjunto de eventos, consulta la tabla 2 con los datos del conjunto presentados en la tabla 1.

Por tanto, las dos clases que desarrollamos en esta práctica están enfocadas a la implementación de un índice sobre `UserID` y de un índice sobre `Brand` para facilitar las consultas de la información sobre los campos `UserID` y `Brand` respectivamente.

- **Clase `Pair`:** Esta clase representa un par `<key, position>`. La `key`, un *string*, hace referencia a la clave de búsqueda (`userID` o `Brand`) mientras que la `position`, un *int*, contiene la posición en el conjunto de eventos donde se encuentra el evento con la clave que andamos buscando.
- **Clase `Index`:** Esta clase contiene un array de objetos tipo `Pair`, y dos atributos: el número de entradas activas en el array (`_nEntries`) y un marcador booleano que nos indica si nuestro índice contiene elementos de tipo marca o de tipo usuario (`_onBrand`). La particularidad de nuestro índice es que está ordenado por los valores de la clave y va a contener valores de clave repetidos. La ventaja de la ordenación por la clave es que nos permite hacer búsquedas eficientes sobre el índice, que de otra manera serían muy lentas sobre el `EventSet` completo.

Además, a partir de esta práctica se utilizará la lectura y escritura directamente sobre ficheros, y comenzarán a utilizarse los argumentos de `main` para modificar el funcionamiento del programa. Se puede consultar más información sobre el uso de parámetros del `main` en el Anexo A. La lectura y escritura de archivos viene detallada en el Anexo B, y en el Anexo C se muestra un caso de ejemplo de las clases `Pair` e `Index` sobre un conjunto de datos sencillo.

2.1. Descripción de la práctica

Para ello, se deberá crear un proyecto NetBeans compuesto por los siguientes ficheros que ya deben estar completos de la práctica anterior:

- **`DateTime.h`**
Nuevo método `sameDay`, para comprobar si dos fechas pertenecen al mismo día
- **`DateTime.cpp`**

⁴O visto de otro modo, el nuevo `Pair` se inserta después del último valor de clave igual.

- **Event.h**

Nuevos métodos `write` and `read`, que permiten leer o escribir un evento desde un flujo de entrada o salida.

- **Event.cpp**

- **EventSet.h** Nuevos métodos `write` and `read`, que permiten leer o escribir un conjunto de eventos desde un flujo de entrada o salida.

- **EventSet.cpp**

- **Pair.h**

Revisar y completar las declaraciones de métodos y parámetros para que pueda funcionar correctamente: métodos `const`, parámetros `const` y parámetros por referencia.

- **Pair.cpp**

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

Además se implementará la función externa a la clase `Pair` siguiente:

- `bool equalKey()`; que comprueba si dos objetos de tipo `Pair` son iguales en el campo `Key`.

- **Index.h**

Revisar y completar las declaraciones de métodos y parámetros para que pueda funcionar correctamente: métodos `const`, parámetros `const` y parámetros por referencia.

- **Index.cpp**

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

También merecen especial mención las funciones siguientes externas a la clase `Index`:

- `Index rawFilterIndex()`; Extrae un subconjunto de `Index`, para el que el valor del campo especificado coincida con el valor indicado.
- `Event getEvent()`; Extrae el evento que ocupa la posición dada en el `Index` dado sobre el `EventSet`.
- `void write()`; Escribe en un fichero los eventos, en formato `csv`, que haya en el índice indicado.
- `float sumPrice()`; Calcula la suma total de las cantidades gastadas (atributo `Price`) en un conjunto de eventos dados por el índice pasado como parámetro.

■ main.cpp

Las lecturas y escrituras se realizarán directamente desde ficheros, en lugar de utilizar la redirección de entrada utilizada en las prácticas anteriores. Se utilizarán los argumentos del main para modificar el comportamiento del programa. En los comentarios del fichero `main.cpp` se describen con detalle los distintos argumentos que puede recibir nuestro programa. Los argumentos que aparecen entre corchetes son argumentos opcionales:

```
-input <input_file> -events <number_events> -index-by UserID|Brand  
[-report-by UserID|Brand|Type|DateTime -output <output_file>  
-display Framed|Fancy]
```

Para poder ejecutar el programa con una de las órdenes anteriores es necesario utilizar la terminal o modificar la orden de ejecución de las propiedades del proyecto (**Run - Run Command**, igual que se hacía para el redireccionamiento de la entrada). Por ejemplo, Si queremos utilizar el archivo de entrada `Ecommerce926.dataset`, leer 500 registros y hacer un índice por marcas, tendríamos que especificar lo siguiente:

```
"${OUTPUT_PATH}" -input ./data/ECommerce926.dataset  
-events 500 -index-by Brand
```

Además habrá que implementar las funciones siguientes:

- `void errorBreak()` ; Que detiene el programa si ocurre algún error (con los argumentos del programa, al abrir un archivo o al leer los datos del fichero)
- `void extractEventSet()` ; Que dado el conjunto de eventos completo y un índice filtrado, crea un nuevo conjunto de eventos sólo con los eventos que están en el índice.

Y completar el código de la función `main` para que el programa tenga el comportamiento que se describe en los comentarios:

1. Procesar los argumentos del main: archivo de entrada, número de eventos a leer, campo sobre el que se realiza el índice (Brand o UserID), campo sobre el que hacer el informe (Brand, UserID, Type o DateTime, usando Brand por defecto ya que es un parámetro opcional), y archivo de salida (también opcional).
2. Utilizar un `EventSet` para almacenar una secuencia de eventos leídos desde el fichero de entrada indicado, leyendo el número de eventos especificados como argumento de entrada (observad que ahora el fichero de eventos no contiene el número de eventos al principio, si no que se especifica desde fuera).
3. Construir el índice sobre las marcas o los usuarios según corresponda.

4. Generar los informes sobre el EventSet siguientes (podéis utilizar la librería DataTable adjunta a la práctica de igual forma que en la práctica 2):
 - Actividad por cada día de la semana (al igual que en las prácticas anteriores, utilizando la función `computeActivity`).
 - Tráfico agregado por el campo indicado con el argumento `-report-by` (suponiendo que agregamos por marca, recuento del número de Eventos para cada marca en el EventSet). Además, almacenar el valor del campo que genera más tráfico, el número de eventos en los que está involucrado, y el índice que contiene dichos eventos, ya que lo necesitaremos más adelante.
 - Precio agregado por el campo indicado con el argumento `-report-by` (suponiendo que agregamos por marca, suma total de los Eventos para cada marca en el EventSet). Para imprimir la información de los distintos DataVector, hay tres funciones disponibles, que se utilizarán en función del parámetro de main `-display`:
 - `dataVector.showPlainReportV(0)` muestra el tipo de informe más simple, y es el que se utiliza por defecto
 - `dataVector.showFramedReportV(0)` muestra un informe de tipo tabla de texto, y es el que se utiliza cuando el parámetro display toma el valor `Framed`
 - `dataVector.showFancyReportV(0)` muestra un informe de tipo gráfico, y es el que se utiliza cuando el parámetro display toma el valor `Fancy`
5. Si se ha especificado un archivo de salida, guardar el eventSet que corresponde al valor del campo que más tráfico ha generado en dicho fichero.
6. Utilizando la función `REPORT_DATA2`, se reportan los siguientes informes para los tests de integración:
 - EventSet original, con todos los eventos leídos del archivo de datos
 - EventSet filtrado, que sólo contiene los eventos indexados en el Index principal construido
 - DataVector con el informe de tráfico generado
 - DataVector con el informe de precios generado

2.2. Un caso de ejemplo

En esta práctica los ficheros en `/tests/validation/` contendrán los parámetros que serán necesarios para que el main pueda utilizar los distintos ficheros de datos que estarán en `/data/`. Así por ejemplo para poder leer del fichero `ECommerce926.dataset`, que contiene 926 eventos, 500 eventos y hacer un índice por marcas se deberá especificar en la redirección de entrada lo siguiente:



```
"${OUTPUT_PATH}" -input ./data/ECommerce926.dataset
-events 500 -index-by Brand
```

Por lo que se podría comprobar que la salida obtenida sería:

```
Read 500 records from file ./data/ECommerce926.dataset
Main Index by Brand
299 pairs
201 records could not be indexed
amount :4926.5
```

Donde se indica que se han leído 500 eventos de `ECommerce926.dataset` y se ha construido un índice por marcas (`Brand`). Sin embargo no todos los eventos contienen una marca por lo que el índice solo contiene 299 pares (se podría utilizar el método `size` de `Index`) y 201 eventos no han sido indexados. El último número nos indica la suma de cantidades (`Price`) de los eventos indexados.

Si calculamos la actividad por día de la semana para los eventos indexados (utilizando `computeActivity()`), después visualizaremos cuántos registros hay en cada día de la semana, lo que produce el siguiente informe:

```
Activity found: SUNDAY(11) MONDAY(20) TUESDAY(73) WEDNESDAY(154)
THURSDAY(17) FRIDAY(23) SATURDAY(1)
```

Si utilizamos un `DataVector` al que llamaremos `weeklyData`, al igual que hicimos en la práctica `shopping2` podremos usar las funciones `showFancyReportH`, `showFramedReportH` o `showPlainReportH` para obtener un informe como el siguiente:

```
Events/day
   11      20      73     154      17      23       1     299
SUNDAY  MONDAY  TUESDAY WEDNESDAY THURSDAY  FRIDAY  SATURDAY
```

donde además se indica el total de eventos (299) procesados.

A continuación, se usa la función `findUnique()` para extraer los valores únicos de la columna `Brand`. Se puede ver que son todos distintos ya que los que no tenían valor no se habían indexado:

```
art-visage  artex  beautix  ...  uskusi  yoko  zinger
```

A continuación se extrae, por cada marca de las anteriores encontradas, el número de veces que aparece cada marca (tráfico por marca) y se suman los precios de los productos que aparecen en cada marca, lo que produce los siguientes dos informes, que llamaremos `amount Data` y `hitsData`, de nuevo usando el tipo de datos `DataVector`. En total, se han vendido 4926.50 \$.

```
Brand (traffic)
art-visage      2
  artex         1
  beautix       2
```



```
.  
. .  
uskusi 5  
yoko 1  
zinger 3  
  
299
```

```
Brand (amount)  
art-visage 9.36  
artex 15.08  
beautix 31.42  
.  
.  
.  
uskusi 19.05  
yoko 7.83  
zinger 48.15  
  
4926.50
```

Además calcularemos qué marca es la que ha conseguido un mayor tráfico (número de visitas) en la web:

```
Highest traffic choice (runail) with 45 hits
```

Finalmente, el programa llama a `REPORT_DATA2` con cada variable interesada, produciendo la salida siguiente, la cual muestra el nombre de la variable, el número de elementos que contiene y el código encriptado de su contenido para que las herramientas de testeo puedan comprobar si se han hecho bien las operaciones internas.

```
Highest traffic choice (runail) with 45 hits  
  
[originalES(Original read from dataset)] 500 18248242769591794958  
  
[filteredES(Events indexed)] 299 3323149042970569627  
  
[hitsData( Report traffic)] 56 art-visage( 2.00) artex( 1.00) beautix( 2.00)  
beauty-free( 4.00) beautyblender( 4.00) bioaqua( 7.00) bpw.style( 1.00)  
browxenna( 2.00) cnd( 4.00) consly( 5.00) depilflax( 1.00) dermal( 4.00)  
domix( 1.00) emil( 1.00) enas( 1.00) enigma( 1.00) estel( 5.00) estelare( 1.00)  
eunyul( 5.00) farmona( 3.00) farmstay( 1.00) grattol( 6.00) haruyama(12.00)  
i-laq( 4.00) ingarden(27.00) irisk(16.00) italwax( 2.00) jas( 2.00)  
jessnail( 1.00) kaaral( 6.00) kapous( 6.00) keen( 2.00) kinetics( 2.00)  
kosmekka( 4.00) lebelage( 1.00) lianail( 1.00) lovely( 5.00) lowence( 4.00)  
marathon( 7.00) masura(14.00) max(11.00) milv( 1.00) oniq(16.00) osmo( 1.00)  
pnb( 2.00) provoc( 3.00) runail(45.00) shik( 2.00) solomeya( 1.00) strong( 6.00)  
sunuv( 1.00) thuya( 8.00) uno(15.00) uskusi( 5.00) yoko( 1.00) zinger( 3.00)  
  
[ammountData( Report ammount)] 56 art-visage( 9.36) artex(15.08) beautix(31.42)  
beauty-free(18.80) beautyblender(99.04) bioaqua(54.46) bpw.style( 0.79)  
browxenna(29.68) cnd(34.28) consly(23.50) depilflax(10.54) dermal( 4.12)  
domix( 4.24) emil(55.56) enas( 3.54) enigma( 6.90) estel(27.45) estelare( 3.02)  
eunyul(35.55) farmona(80.73) farmstay( 0.63) grattol(25.14) haruyama(47.64)  
i-laq(15.88) ingarden(174.99) irisk(129.85) italwax(26.04) jas(51.42)  
jessnail( 8.73) kaaral(59.22) kapous(21.90) keen(19.56) kinetics(18.74)  
kosmekka(128.88) lebelage( 3.75) lianail( 6.33) lovely(34.93) lowence(69.84)  
marathon(964.46) masura(57.88) max(579.86) milv( 3.17) oniq(139.68) osmo( 9.71)  
pnb( 7.38) provoc(19.71) runail(63.61) shik(20.64) solomeya( 1.75)  
strong(1172.22) sunuv(33.33) thuya(252.06) uno(134.48) uskusi(19.05) yoko( 7.83)  
zinger(48.15)
```


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 MPTTest

([Abrir en navegador →](#))

Esta tercera práctica mantiene los tres niveles de testeo, que incluyen tanto tests unitarios como de integración y que incluyen los tests de la práctica anterior más los nuevos tests de esta práctica (Ver Anexo ??).

- Nivel Básico: 27 tests
- Nivel Intermedio: 10 tests
- Nivel Avanzado: 22 tests. Varios de ellos hacen un chequeo de memoria adicional, para comprobar que ningún vector de datos se sale más allá de sus límites.

Y este debería ser el resultado del test de la aplicación satisfaciendo todos los tests que se han diseñado (en azul aparece la llamada a los tests desde una terminal del proyecto).

```
make test

[=====] Running 59 tests from 3 test suites.
[-----] Global test environment set-up.
[-----] 27 tests from _01_Basics
[ RUN      ] _01_Basics.DateTime_Constructors
[ OK       ] _01_Basics.DateTime_Constructors (5 ms)
[ RUN      ] _01_Basics.DateTime_getters
[ OK       ] _01_Basics.DateTime_getters (6 ms)
[ RUN      ] _01_Basics.DateTime_set
[ OK       ] _01_Basics.DateTime_set (2 ms)
[ RUN      ] _01_Basics.DateTime_sameDay
[ OK       ] _01_Basics.DateTime_sameDay (5 ms)
[ RUN      ] _01_Basics.Event_ConstructorBase
[ OK       ] _01_Basics.Event_ConstructorBase (4 ms)
[ RUN      ] _01_Basics.Event_Setters_getters
[ OK       ] _01_Basics.Event_Setters_getters (9 ms)
[ RUN      ] _01_Basics.EventSet_Constructor
[ OK       ] _01_Basics.EventSet_Constructor (3 ms)
[ RUN      ] _01_Basics.EventSet_add_event
[ OK       ] _01_Basics.EventSet_add_event (5 ms)
[ RUN      ] _01_Basics.EventSet_add_line
[ OK       ] _01_Basics.EventSet_add_line (4 ms)
[ RUN      ] _01_Basics.EventSet_at_basic
[ OK       ] _01_Basics.EventSet_at_basic (4 ms)
[ RUN      ] _01_Basics.Pair_Constructors
[ OK       ] _01_Basics.Pair_Constructors (4 ms)
[ RUN      ] _01_Basics.Pair_isEmpty
[ OK       ] _01_Basics.Pair_isEmpty (2 ms)
[ RUN      ] _01_Basics.Pair_setters
[ OK       ] _01_Basics.Pair_setters (1 ms)
[ RUN      ] _01_Basics.Pair_getters
[ OK       ] _01_Basics.Pair_getters (0 ms)
[ RUN      ] _01_Basics.Index_Constructors
[ OK       ] _01_Basics.Index_Constructors (1 ms)
[ RUN      ] _01_Basics.Index_getIONWhich
[ OK       ] _01_Basics.Index_getIONWhich (1 ms)
[ RUN      ] _01_Basics.Index_clear
[ OK       ] _01_Basics.Index_clear (2 ms)
[ RUN      ] _01_Basics.Integrated_5_records
[ MEMCHECK ] ECommerce5-valgrind
[ OK       ] ECommerce5-valgrind
[ OK       ] _01_Basics.Integrated_5_records (28 ms)
[ RUN      ] _01_Basics.Integrated_30_records
[ MEMCHECK ] ECommerce30-valgrind
[ OK       ] ECommerce30-valgrind
[ OK       ] _01_Basics.Integrated_30_records (24 ms)
[ RUN      ] _01_Basics.Integrated_41_records
[ MEMCHECK ] ECommerce41-valgrind
[ OK       ] ECommerce41-valgrind
[ OK       ] _01_Basics.Integrated_41_records (31 ms)
[ RUN      ] _01_Basics.Integrated_162_records
[ MEMCHECK ] ECommerce162-valgrind
[ OK       ] ECommerce162-valgrind
[ OK       ] _01_Basics.Integrated_162_records (37 ms)
[ RUN      ] _01_Basics.Integrated_926_records
[ MEMCHECK ] ECommerce926-valgrind
```




```
[ OK ] ECommerce926-valgrind
[ OK ] _01_Basics.Integrated_926_records (139 ms)
[ RUN ] _01_Basics.Integrated_Args_5_records
[ MEMCHECK ] ECommerce5-valgrind
[ OK ] ECommerce5-valgrind
[ OK ] _01_Basics.Integrated_Args_5_records (27 ms)
[ RUN ] _01_Basics.Integrated_Args_30_records
[ MEMCHECK ] ECommerce30-valgrind
[ OK ] ECommerce30-valgrind
[ OK ] _01_Basics.Integrated_Args_30_records (28 ms)
[ RUN ] _01_Basics.Integrated_Args_41_records
[ MEMCHECK ] ECommerce41-valgrind
[ OK ] ECommerce41-valgrind
[ OK ] _01_Basics.Integrated_Args_41_records (28 ms)
[ RUN ] _01_Basics.Integrated_Args_162_records
[ MEMCHECK ] ECommerce162-valgrind
[ OK ] ECommerce162-valgrind
[ OK ] _01_Basics.Integrated_Args_162_records (29 ms)
[ RUN ] _01_Basics.Integrated_Args_926_records
[ MEMCHECK ] ECommerce926-valgrind
[ OK ] ECommerce926-valgrind
[ OK ] _01_Basics.Integrated_Args_926_records (134 ms)
[-----] 27 tests from _01_Basics (563 ms total)

[-----] 10 tests from _02_Intermediate
[ RUN ] _02_Intermediate.DateTime_isBefore
[ OK ] _02_Intermediate.DateTime_isBefore (4 ms)
[ RUN ] _02_Intermediate.DateTime_weekDay
[ OK ] _02_Intermediate.DateTime_weekDay (4 ms)
[ RUN ] _02_Intermediate.Event_getField
[ OK ] _02_Intermediate.Event_getField (4 ms)
[ RUN ] _02_Intermediate.EventSet_add_event_partial
[ OK ] _02_Intermediate.EventSet_add_event_partial (4 ms)
[ RUN ] _02_Intermediate.EventSet_at_intermediate
[ OK ] _02_Intermediate.EventSet_at_intermediate (5 ms)
[ RUN ] _02_Intermediate.Index_3x3_just_build
[ OK ] _02_Intermediate.Index_3x3_just_build (4 ms)
[ RUN ] _02_Intermediate.Index_B_BxU_build_at
[ OK ] _02_Intermediate.Index_B_BxU_build_at (7 ms)
[ RUN ] _02_Intermediate.Index_U_BxU_build_at
[ OK ] _02_Intermediate.Index_U_BxU_build_at (4 ms)
[ RUN ] _02_Intermediate.Integrated_EMPTY
[ MEMCHECK ] EMPTY-valgrind
[ OK ] EMPTY-valgrind
[ OK ] _02_Intermediate.Integrated_EMPTY (44 ms)
[ RUN ] _02_Intermediate.Integrated_ErrorLoading
[ MEMCHECK ] ErrorLoading-valgrind
[ OK ] ErrorLoading-valgrind
[ OK ] _02_Intermediate.Integrated_ErrorLoading (40 ms)
[-----] 10 tests from _02_Intermediate (121 ms total)

[-----] 22 tests from _03_Advanced
[ RUN ] _03_Advanced.DateTime_BadValues
[ OK ] _03_Advanced.DateTime_BadValues (8 ms)
[ RUN ] _03_Advanced.Event_setType_Bad_Values
[ OK ] _03_Advanced.Event_setType_Bad_Values (3 ms)
[ RUN ] _03_Advanced.Event_Others_Bad_Values
[ OK ] _03_Advanced.Event_Others_Bad_Values (4 ms)
[ RUN ] _03_Advanced.EventSet_add_event_full
[ OK ] _03_Advanced.EventSet_add_event_full (6 ms)
[ RUN ] _03_Advanced.EventSet_at_advanced
[ OK ] _03_Advanced.EventSet_at_advanced (4 ms)
[ RUN ] _03_Advanced.EventSet_externalfunctions
[ OK ] _03_Advanced.EventSet_externalfunctions (5 ms)
[ RUN ] _03_Advanced.EventSet_write
[ OK ] _03_Advanced.EventSet_write (3 ms)
[ RUN ] _03_Advanced.EventSet_read
[ OK ] _03_Advanced.EventSet_read (2 ms)
[ RUN ] _03_Advanced.Index_U_BxU_bounds
[ OK ] _03_Advanced.Index_U_BxU_bounds (1 ms)
[ RUN ] _03_Advanced.Index_B_BxU_bounds
[ OK ] _03_Advanced.Index_B_BxU_bounds (1 ms)
[ RUN ] _03_Advanced.Index_add
[ OK ] _03_Advanced.Index_add (2 ms)
[ RUN ] _03_Advanced.Index_B_BxU_rawFilterIndex
[ OK ] _03_Advanced.Index_B_BxU_rawFilterIndex (1 ms)
[ RUN ] _03_Advanced.Index_U_BxU_rawFilterIndex
[ OK ] _03_Advanced.Index_U_BxU_rawFilterIndex (2 ms)
[ RUN ] _03_Advanced.Index_Type_BxU_rawFilterIndex
[ OK ] _03_Advanced.Index_Type_BxU_rawFilterIndex (2 ms)
[ RUN ] _03_Advanced.Index_DateTime_BxU_rawFilterIndex
[ OK ] _03_Advanced.Index_DateTime_BxU_rawFilterIndex (2 ms)
[ RUN ] _03_Advanced.Index_BxU_sumPrice
[ OK ] _03_Advanced.Index_BxU_sumPrice (5 ms)
[ RUN ] _03_Advanced.Integrated_ErrorData
[ MEMCHECK ] ErrorData-valgrind
[ OK ] ErrorData-valgrind
[ OK ] _03_Advanced.Integrated_ErrorData (34 ms)
[ RUN ] _03_Advanced.Integrated_ErrorSaving
[ MEMCHECK ] ErrorSaving-valgrind
[ OK ] ErrorSaving-valgrind
[ OK ] _03_Advanced.Integrated_ErrorSaving (36 ms)
[ RUN ] _03_Advanced.Integrated_Args_no_open
[ MEMCHECK ] ECommerce926-valgrind
[ OK ] ECommerce926-valgrind
[ OK ] _03_Advanced.Integrated_Args_no_open (34 ms)
[ RUN ] _03_Advanced.Integrated_Args_error_data
[ MEMCHECK ] ECommerce162-valgrind
[ OK ] ECommerce162-valgrind
[ OK ] _03_Advanced.Integrated_Args_error_data (38 ms)
```

```
[ RUN      ] _03_Advanced.Integrated_Args_error_missing_arg
[ MEMCHECK ] ECommerce162-valgrind
[ OK       ] ECommerce162-valgrind
[ MEMCHECK ] ECommerce162-valgrind
[ OK       ] ECommerce162-valgrind
[ OK       ] _03_Advanced.Integrated_Args_error_missing_arg (61 ms)
[ RUN      ] _03_Advanced.Integrated_Args_error_bad_arg
[ MEMCHECK ] ECommerce162-valgrind
[ OK       ] ECommerce162-valgrind
[ OK       ] _03_Advanced.Integrated_Args_error_bad_arg (30 ms)
[-----] 22 tests from _03_Advanced (288 ms total)

[-----] Global test environment tear-down
[=====] 59 tests from 3 test suites ran. (972 ms total)
[ PASSED ] 59 tests.
```

2.4. Configuración de la práctica

Para esta práctica se puede seguir con la misma configuración de la práctica anterior pero con las siguientes novedades:

- Se incluirán en la carpeta **tests** los nuevos tests correspondientes a las clases **Pair** e **Index**, así como los nuevos tests de integración.
- Se incluirán los ficheros correspondientes a las clases **Pair** e **Index** a las carpetas **include** y **src**.
- Se deberá añadir la carpeta **include** de la librería **DataTable** mediante **Project Properties - C++ Compiler - Include Directories**. Opcionalmente se podrá añadir también la librería **AnsiTerminal**, incluyendo también su carpeta **include**. Revisar que ya están también incluidos los directorios **include** siguientes:
 - del propio proyecto **./include**
 - las de testeo **../MPTest/include**
 - las de googletest **../googletest-master/googletest/include**
- Se modificará el fichero **main.cpp** de acuerdo a las instrucciones en la documentación del fichero de la práctica.
- Recordar añadir desde la vista lógica del proyecto.
 1. En **Header Files, Add existing item** añadir los ficheros **Pair.h** e **Index.h**.
 2. En **Source Files, Add existing item** añadir los ficheros **Pair.cpp** e **Index.cpp**.
 3. En **Test Files, Add existing item** añadir los nuevos ficheros de los tests.

La práctica deberá ser entregada en Prado, en la fecha que se indica en cada entrega, y consistirá en un fichero ZIP del proyecto en su estado actual (es decir con todas las clases implementadas) incluyendo las clases implementadas para la práctica anterior.



2.5. Entrega de la práctica

Una vez terminada la práctica que, al menos, haya superado los tests básicos, se debe hacer un zip (se sugiere utilizar la script `runZipProject.sh`) excluyendo las carpetas `./dist/`, `./build/`, `./nbproject/private/`, `./doc/html/` y `./dos/latex/` y subirla a Prado antes de la fecha de cierre de la entrega.



3. TESTS DOCUMENTATION FOR PROJECT Shopping3

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. DateTime_sameDay

1. SKIP_ASSERT_TRUE_R(dt1.sameDay(dt3));
2. SKIP_ASSERT_FALSE_R(dt1.sameDay(dt2));

3.1.5. Event_ConstructorBase

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

3.1.6. 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



3.1.7. **EventSet_Constructor**

1. A newly created instance of EventSet must have size = 0
2. A newly created instance of EventSet must have a to_string empty '0'

3.1.8. **EventSet_add_event**

1. Adding an event to a newly created instance of EventSet must have size = 1
2. Adding the default event to an empty event set must have a to_string equal to the default event
3. Adding 1 event to a filled EventSet increases its size in 1

3.1.9. **EventSet_add_line**

1. Adding a line to a newly created instance of EventSet must have size = 1
2. Adding the default event, as a line, to an empty event set must have a to_string equal to the default event
3. Adding 1 event, as a line, to a filled EventSet increases its size in 1

3.1.10. **EventSet_at_basic**

1. Querying the event at the 0 position should match with the first event added to the EventSet
2. Querying the event at the middle position should match with the event added which was added at that point

3.1.11. **Pair_Constructors**

1. After the creation of a new instance of Pair, it shows its default value
2. After the creation of a new instance of Pair by using a string and an int, it shows these two same values
3. After the creation of a new instance of Pair by using a string and NO int, it shows the string and -1 default value

3.1.12. **Pair_isEmpty**

1. Any newly created instance of Pair, without specifying any arguments, is considered to be empty
2. Any instance of Pair with valid key and pos, could not be considered empty

3.1.13. **Pair_setters**

1. After the creation of a new instance of Pair and setting it by using a string and an int, it shows these two same values
2. After the creation of a new instance of Pair and setting the key to a given string, it shows this same string



3. After the creation of a new instance of Pair and setting the value to a given number, it shows this same number

3.1.14. Pair_getters

1. After the creation of a new instance of Pair and setting it by using a string and an int, getKey() returns the same string
2. After the creation of a new instance of Pair and setting it by using a string and an int, getPost() returns the same integer
3. After the creation of a new instance of Pair and setting the key to a given string, getKey() returns this same string
4. After the creation of a new instance of Pair and setting the value to a given number, getPos() returns the same number

3.1.15. Index_Constructors

1. A newly created instance of Index must have size = 0
2. A newly created instance of Index must write as "": On brand? 0"

3.1.16. Index_getlOnWhich

1. Building an index by Users reports 0
2. Building an index by brand reports 1

3.1.17. Index_clear

1. Clearing a brand new instance of Index leaves it empty
2. Clearing an instance of Index with existing records leaves it empty

3.1.18. Integrated_5_records

1. Execution of a simple test with five records only

3.1.19. Integrated_30_records

1. Execution of a test with 30 records only

3.1.20. Integrated_41_records

1. Execution of a test with 41 records only

3.1.21. Integrated_162_records

1. Execution of a test with 162 records only

3.1.22. Integrated_926_records

1. Execution of a test with 926 records only



3.1.23. Integrated_Args_5_records

1. Execution of a simple test with five records using main arguments

3.1.24. Integrated_Args_30_records

1. Execution of a test with 30 records using main arguments

3.1.25. Integrated_Args_41_records

1. Execution of a test with 41 records using main arguments

3.1.26. Integrated_Args_162_records

1. Execution of a test with 162 records using main arguments

3.1.27. Integrated_Args_926_records

1. Execution of a test with 926 records using main arguments

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

3.2.3. Event_getField

1. getField("DateTime") on any Event, must be equal to getDateTime()
2. In any row, querying the Event for field "Price" will produce the same string than getDateTime
3. In any case, querying the Event for field "UserID" will produce the same string than the former getUserID()
4. In any row, querying the EventS a mandatory field cannot give an empty string
5. In any row, querying an optional field will produce the same string than the former event added to the EvenSet



3.2.4. EventSet_add_event_partial

1. Adding MAXEVENT events to a newly created EventSet increases its size in MAXEVENT

3.2.5. EventSet_at_intermediate

1. Querying the event at the last position should match with the last event added to the EventSet
2. Accessing EventSet at() a certain position and changing the user ID of the event, this change will remain in the EventSet itself
3. Accessing EventSet at() a certain position and changing the brand of the event, this change will remain in the EventSet itself

3.2.6. Index_10x10_just_build

1. In any case, building the same index twice gives the same result

3.2.7. Index_B_BxU_build_at

1. The index by brand built within experiment BxU is the same key during U consecutive positions
2. The index by brand built within experiment BxU two consecutive indexes of the same brand differ in B units
3. The index by user built within experiment BxU changes the key at point U

3.2.8. Index_U_BxU_build_at

1. The index by user built within experiment BxU is the same key during B consecutive positions
2. The index by user built within experiment BxU indexes B consecutive positions for the same key
3. The index by user built within experiment BxU changes the key at point B

3.2.9. Integrated_EMPTY

1. Execution of a simple test with an empty set of records

3.2.10. Integrated_ErrorLoading

1. Execution of a simple test with an error when opening the data file

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

3.3.4. EventSet.add_event_full

1. Adding one single Event events to a partly filled EventSet must return 1
2. Adding one single Event events to a completely filled EventSet does not produce any change in the EventSet
3. Adding one single Event events to a completely filled EventSet must return 0

3.3.5. EventSet.at_advanced

1. Querying an EventSet beyond its legal limits gives an EMPTY event
2. Querying an EventSet beyond its legal limits gives an EMPTY event
3. Querying an EventSet within its legal limits always gives a NON-EMPTY event

3.3.6. EventSet.externalfunctions

1. sumPrice must give the sum of all prices of a given EventSet, and it doesnt
2. UniqueBrands() must give the number of different brands, but it doesnt
3. UniqueUsers() must give the number of different users, but it doesnt

3.3.7. EventSet.write

1. Any EventSet writes into a disk file exactly its to_string() without the number of events



3.3.8. **EventSet_read**

1. Any EventSet read from a disk file must have a `to_string()`, without the number of events, comparable to the content of the file

3.3.9. **Index_U_BxU_bounds**

1. The index by user built within experiment BxU lower bounds User u with $u*B$
2. The index by brand built within experiment BxU upper bounds Brand b with $(b+1)*U$

3.3.10. **Index_B_BxU_bounds**

1. The index by brand built within experiment BxU lower bounds Brand b with $b*U$
2. The index by brand built within experiment BxU upper bounds Brand b with $(b+1)*U$

3.3.11. **Index_add**

1. Inserting any pair in any index increases its size in one
2. Adding a key at the end of the index places the new key in its previous upperbound
3. Adding a key at the beginning of the index gives a lower bound equals to 0 after the insertion

3.3.12. **Index_B_BxU_rawFilterIndex**

1. In an Index BxU , filtering the index by brand- i produces an index with exactly size U

3.3.13. **Index_U_BxU_rawFilterIndex**

1. In an Index BxU , filtering the index by user- i produces an index with exactly size B

3.3.14. **Index_Type_BxU_rawFilterIndex**

1. In an Index BxU , filtering the index by Type=cart produces an index with exactly size B

3.3.15. **Index_DateTime_BxU_rawFilterIndex**

1. In an Index BxU , filtering the index by DateTime equals to the first available date produces an index with exactly size B



3.3.16. Index_BxU_sumPrice

1. Given a BxU Index filtered by user u , the sum of prices amounts to $(uB)/4(1+B)/2$
2. Given a BxU Index filtered by brand b , the sum of prices amounts to $U*(b+1)$
3. Given a BxU Index filtered by type view, the sum of prices amounts to $(BU)/4(1+B)/2$
4. Given a BxU Index filtered by datetime day , the sum of prices amounts to $U*day$

3.3.17. Integrated_ErrorData

1. Execution of a simple test with an error when reading the data file

3.3.18. Integrated_ErrorSaving

1. Execution of a simple test with an error when saving the data file

3.3.19. Integrated_Args_no_open

1. Execution of a simple test with main args where the input file is not found

3.3.20. Integrated_Args_error_data

1. Execution of a simple test with main args where the number of specified events is greater than the total number of events in the data file

3.3.21. Integrated_Args_missing_arg

1. Execution of a test with main args where the input file is not specified

3.3.22. Integrated_Args_error_bad_arg

1. Execution of a test with main args where one of the arguments is not understood

A. Argumentos de `main`

Hasta el momento, los programas que hemos implementado tenían siempre la misma funcionalidad, y para poder cambiar su comportamiento teníamos que reescribir el código y volver a compilar, o interactuar con él usando el teclado durante su ejecución. No obstante, la interacción con el programa a través del teclado puede no ser la forma más adecuada de cambiar su comportamiento en determinados contextos. Por ejemplo, el comando `ls`, de la terminal de Linux que ya debes conocer (el cual es un programa escrito en C), nos permite mostrar el contenido del directorio que se pasa como argumento. Si no se pasa ningún argumento, se muestra el contenido del directorio actual:

```
fluque1995@fluque1995-Aspire-VX5-5916:~/Netbeans$ ls
DataTable      Gtest          Scripts         Shopping2_S
DataTable.zip  MPTTest        Shopping1_S     Shopping2_S_files
googletest-master MPTTest_V2.zip Shopping2       Shopping2_S.zip
fluque1995@fluque1995-Aspire-VX5-5916:~/Netbeans$ ls Shopping1_S
build      dist      Makefile  __nTest  src
data       doc       nbproject __nTestname tests
dirInfo.md include  __nReport scripts  zip
```

Figura 5: Ejemplo de ejecución del program `ls` con y sin parámetros

Aquí tenemos un ejemplo claro de un programa cuyo comportamiento depende de los parámetros que recibe. Como podrás imaginar, preguntar al usuario en el momento de la ejecución por el directorio del que se quiere mostrar el contenido puede ser bastante incómodo. Además, el hecho de poder especificarlo de antemano permite que el programa se pueda ejecutar sin necesidad de que una persona interactúe con el programa en el momento de la ejecución.

Los parámetros o argumentos de la función `main` son, precisamente, las herramientas de las que disponemos para cambiar el comportamiento de nuestro programa. Funcionan de la misma forma que los argumentos de cualquier otra función. C++ acepta dos tipos de declaraciones de la función `main`, con y sin argumentos:

```
int main(){
    // Función main sin argumentos
}

int main(int argc, char **argv){
    // Función main con argumentos
}
```

Como se puede observar, cuando trabajamos con la declaración del `main` con argumento, se reciben dos argumentos distintos:

- `int argc`: Entero que indica el número de argumentos que se han recibido
- `char **argv`: Array de cadenas de caracteres con cada uno de los argumentos recibidos

A continuación se muestra un ejemplo de programa que recorre todos los argumentos que recibe y los imprime por pantalla:

```
#include <iostream>

using namespace std;
int main(int argc, char** argv) {

    /* Usage of main arguments. Main receives a series of arguments when called from the terminal:
     * - argc: Number of arguments passed
     * - argv: Array of strings representing the arguments
     */
    cout << "Number_of_arguments_passed_to_this_program:_" << argc << endl;

    /* argv[0] is always the name of the program
     */
    cout << "Name_of_the_program:_" << argv[0] << endl;

    /* The rest of the arguments (if passed), come ordered. We use argc as stopping point (the size of argv)
     * We start from 1 since argv[0] is the name of the program
     */
    if (argc == 1){
        cout << "No_more_arguments_passed" << endl;
    }
    for (int i = 1; i < argc; i++){
        cout << "Argument_" << i << ":_" << argv[i] << endl;
    }
    return 0;
}
```

Y la ejecución del programa depende ahora de la llamada que hagamos desde la terminal:

```
~/N/M/d/D/GNU-Linux > ./mainarguments
Number of arguments passed to this program: 1
Name of the program: ./mainarguments
No more arguments passed
~/N/M/d/D/GNU-Linux > ./mainarguments Hola
Number of arguments passed to this program: 2
Name of the program: ./mainarguments
Argument 1: Hola
~/N/M/d/D/GNU-Linux > ./mainarguments Hola mundo
Number of arguments passed to this program: 3
Name of the program: ./mainarguments
Argument 1: Hola
Argument 2: mundo
```

Figura 6: Ejemplos de ejecución del programa anterior con distintos argumentos

De esta manera, podremos modificar el comportamiento de nuestro programa sin necesidad de interactuar con él en el momento de la ejecución, o sin tener que recompilarlo.

En nuestro proyecto, será de gran utilidad poder emplear los argumentos de main. Por ejemplo, en la práctica Shopping2 creamos un primer filtro sobre el conjunto de eventos, que nos permitía seleccionar sólo aquellos eventos que eran de tipo compra.

Si hubiésemos querido filtrar nuestro conjunto de datos para seleccionar sólo aquellos eventos que fueran de tipo view, en lugar de purchase, habríamos tenido que modificar el código del programa y volver a compilarlo. Ahora, utilizando los parámetros de main, podremos especificar el tipo de evento por el que filtramos. En el archivo `main.cpp` de la práctica aparece información sobre los distintos argumentos que habrá que procesar, y que condicionarán el comportamiento de nuestro programa (el archivo de entrada, número de líneas que hay que leer del mismo, valores por los que querremos filtrar, etc...).

B. La biblioteca `fstream`

En las prácticas anteriores, hemos simulado que los datos se leían desde el teclado utilizando el operador de redirección de entrada (símbolo `<` en la terminal) cuando era necesario. La llamada a nuestros programas desde la terminal tenía, más o menos, este aspecto:

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

Y dentro del código de nuestro programa leíamos desde `cin` cuando era necesario. El inconveniente de este tipo de lectura es que nos permite poca flexibilidad. Por ejemplo, no podemos leer información de entrada y escribir información de salida en varios ficheros separados. Por tanto, es conveniente disponer de un mecanismo que nos permita leer y escribir información directamente sobre ficheros, en lugar de leer únicamente del teclado (flujo de entrada `cin`) y escribir por pantalla (flujo de salida `cout`). Para ello, tenemos la librería `fstream` (se puede consultar su documentación en <http://www.cplusplus.com/reference/fstream/fstream/>).

El modo normal en que procesamos un archivo de texto es leerlo/escribirlo desde el principio hasta el final, por lo que hablaremos de flujos de caracteres (stream). El concepto de flujo de entrada/salida permite ocultar detalles sobre cómo funcionan los dispositivos externos. El programador considera el flujo como una secuencia de caracteres que puede leer o escribir, sin preocuparse de cómo se produce realmente ese intercambio de información con los archivos. La conexión con el fichero se hace a través de una zona de memoria conocida como buffer donde se ubican de forma temporal los caracteres a ser tratados. Esto permite minimizar los accesos a disco (más lentos), ya que en cada lectura física de disco se carga todo lo posible en el buffer de entrada y las escrituras no se producen físicamente hasta que el buffer de salida está completo. Además, también nos independiza de cómo el sistema operativo realiza dichas operaciones de carga y descarga de los buffers.

En general, cuando leemos un fichero de texto, nos ubicamos en el principio del mismo, y vamos procesando los caracteres en orden hasta llegar al final del archivo. De igual modo, cuando escribimos, se empieza a escribir en el principio del archivo y se vuelcan en orden todos y cada uno de los caracteres al disco. En ambos casos se hace un recorrido secuencial sobre los ficheros, siendo la forma más habitual para tratar con ficheros de entrada/salida. En ningún momento intentamos posicionarnos de forma directa en posiciones intermedias del archivo, por ejemplo en el carácter 1526.

B.1. Primeros Pasos

El estándar C++ permite hacer la entrada/salida de datos de una forma muy similar a como se realiza la entrada/salida desde consola con `cin` y `cout`. Recordemos que `cin` es un objeto de la clase `istream`

(input stream, flujo de entrada) y `cout` es un objeto de la clase `ostream` (output stream, flujo de salida). Tanto `cin` como `cout` son objetos globales creados en el estándar C++, por lo que los tenemos a disposición del programador al incluir `iostream`.

Cuando trabajamos con archivos de texto tenemos definidas dos clases con un comportamiento similar: la clase `ifstream` (flujo de archivos de entrada) hereda de `istream`, y la clase de `ofstream` (flujo de archivos de salida) hereda de `ostream`. Por lo tanto, todas las funciones miembro y operadores que se pueden utilizar con objetos `istream` u `ostream` también se pueden aplicar a los objetos de la clase `ifstream` y `ofstream`. No obstante, hay una primera diferencia: Es el propio programador el que se debe encargar de crear el objeto y asociarlo con un determinado fichero de entrada/salida. Esto nos permite controlar qué archivos se utilizan y el propósito con el que los utilizamos.

Una segunda diferencia es que para poder trabajar con archivos dispondremos de algunas funciones adicionales e información interna (atributos) que permiten controlar la *conexión* del flujo de entrada con los archivos físicos en disco.

B.2. Abriendo y cerrando ficheros

En primer lugar, debemos de declarar tantos objetos del tipo `fstream` como archivos queramos utilizar de forma simultánea. En el siguiente fragmento de código usamos dos objetos, uno para realizar la entrada (lectura) de datos y otro de salida (escritura). Pero un programa puede tener y usar tantos archivos como desee, para ello debemos declarar un objeto para cada archivo.

```
#include <iostream>
#include <fstream> // library for files manipulation

usando namespace std;
...
int main ()
{
    ifstream input_file; // input stream
    ofstream output_file; // output stream
    ...
}
```

En este caso sólo hemos declarado los objetos, indicando si se utilizarán como flujos entrada o de salida. Para trabajar con los ficheros debemos de poder asociarle a cada flujo el fichero concreto en disco sobre el que queremos trabajar. Para ello, debemos indicarle al objeto de flujo de datos el nombre del archivo (mediante un `string` o `cstring`) y el objeto se encargará de gestionar con el Sistema Operativo las conexiones necesarias para su lectura/escritura. Continuando con el ejemplo:

```
#include <iostream>
#include <fstream> // library for files manipulation

usando namespace std;
...
int main ()
{
    ifstream input_file; // input stream
    input_file.open("input.txt");
    ofstream output_file; // output stream
    output_file.open("output.txt");
    ...
}
```

```
}
```

La apertura de los ficheros puede estar sujeta a problemas, como que el fichero que queremos abrir para lectura no exista (hayamos introducido un nombre de fichero incorrecto), o que no tengamos permisos de escritura sobre el fichero de salida. Para estos casos, es conveniente comprobar que la apertura del fichero ha sido correcta antes de utilizarlo. Para ello, disponemos de la función `is_open()`, la cual devuelve `true` cuando la lectura del fichero ha sido correcta.

```
if (input_file.is_open()){  
    // Correct open of input file , we can read normally  
}  
  
if (input_file.is_open()){  
    // Correct open of output file , we can write normally  
}
```

Una vez hemos comprobado que la apertura del fichero es correcta, podemos utilizar el flujo de entrada y salida normalmente, al igual que utilizábamos los flujos de entrada y salida de terminal (`cin` y `cout`). En particular, tendremos disponibles los operadores `operator>>` para leer del flujo de entrada y `operator<<` para leer del flujo de salida:

```
ifstream input_file; // input stream  
input_file.open("input.txt");  
ofstream output_file; // output stream  
output_file.open("output.txt");  
  
int x;  
if (input_file.is_open()){  
    // Correct open of input file , we read an integer  
    input_file >> x;  
}  
  
if (input_file.is_open()){  
    // Correct open of output file , we write the integer  
    output_file << x;  
}
```

Una vez hemos terminado de trabajar con nuestros archivos, es necesario cerrar sus flujos. Cerrar un archivo implica desconectar el objeto `fstream` del archivo físico y decirle al Sistema Operativo que realice las tareas necesarias para guardar su estado de forma correcta. En el caso de un fichero de salida, su cierre nos garantiza que toda la información que podría estar guardada temporalmente en el buffer de memoria, pero no volcada físicamente al fichero en disco, sea guardada de forma correcta. Recordemos que para ahorrar accesos a discos (bastante costosos), se realizan sólo cuando el buffer esta lleno. Cerrar el fichero garantiza que la información se vuelca de forma correcta.

Para cerrar los archivos, basta con llamar al método `close()`:

```
input_file.close();  
output_file.close();
```

B.3. Ejemplo de uso de archivos

Veamos un pequeño ejemplo funcional que trabaja con archivos de entrada y salida. En particular, tendremos el archivo `input.txt`, que contendrá una secuencia de diez números enteros. Nuestro programa leerá la secuencia de enteros de dicho archivo, los elevará al cuadrado, y los guardará en el archivo de salida `output.txt`. El código del programa es el siguiente:


```
#include <cstdlib>
#include <iostream>
#include <fstream>

using namespace std;

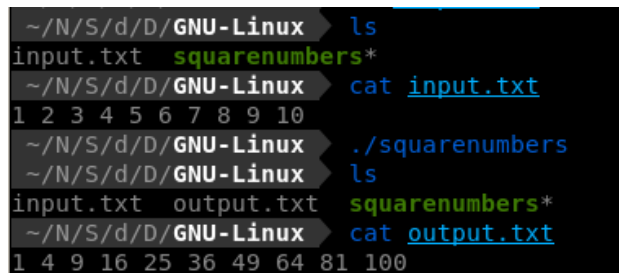
int main(int argc, char** argv) {
    ifstream input_file; // Creation of input stream
    ofstream output_file; // Creation of output stream

    int aux, n_numbers=10; // Aux int and amount of numbers to read

    input_file.open("input.txt"); // Open input stream
    output_file.open("output.txt"); // Open output stream

    if (input_file.is_open() && output_file.is_open()) { // if both are opened correctly
        for (int i = 0; i < n_numbers; i++){
            input_file >> aux; // read from input file
            output_file << aux * aux << " "; // write in output file squaring the number
        }
        output_file << endl;
        // Close both streams at the end
        input_file.close();
        output_file.close();
    }
    return 0;
}
```

Si compilamos el programa y lo ejecutamos desde la terminal, podemos ver cómo se crea el archivo de salida con el contenido correcto:



```
~/N/S/d/D/GNU-Linux ls
input.txt squarenumbers*
~/N/S/d/D/GNU-Linux cat input.txt
1 2 3 4 5 6 7 8 9 10
~/N/S/d/D/GNU-Linux ./squarenumbers
~/N/S/d/D/GNU-Linux ls
input.txt output.txt squarenumbers*
~/N/S/d/D/GNU-Linux cat output.txt
1 4 9 16 25 36 49 64 81 100
```

Figura 7: Ejecución del programa anterior

Como se puede observar en la imagen, en primer lugar tenemos el ejecutable y el archivo de entrada con los números del 1 al 10. Cuando lo ejecutamos, aparece el archivo de salida (como no existe, se crea, en caso de existir por defecto se sobrescribe), y su contenido es justo el esperado, el cuadrado de los números del 1 al 10.

C. Ejemplo de uso de las clases Pair e Index

En esta sección se ejemplifica el uso de las clases Pair e Index. Supongamos que estamos interesados en extraer información sobre el usuario con código 446419295. En particular, estamos interesados en saber qué cantidad de eventos ha generado dicho usuario, así como el precio total de dichos eventos, y trabajamos con el conjunto de datos que se muestra a continuación:



DateTime	EventType	ProductID	CategoryID	Cat Code	Brand	Price	UserID	Session ID
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 00:29:21 UTC	cart	5857007	1487580009496313889		runail	3.17	554342223	0b974342-1a53-41c1-a426-23130e770f4b
2019-10-01 00:36:22 UTC	view	7652	1487580008657454075			2.86	446419295	d5e0977b-b6ac-43a1-b173-63e9742e95b2
2019-10-01 00:37:27 UTC	view	39918	1487580008657454075			2.86	446419295	d5e0977b-b6ac-43a1-b173-63e9742e95b2
2019-10-01 00:38:33 UTC	view	5582544	1958278551207674674			2.7	446419295	d5e0977b-b6ac-43a1-b173-63e9742e95b2
2019-10-01 00:39:45 UTC	view	7652	1487580008657454075			2.86	446419295	d5e0977b-b6ac-43a1-b173-63e9742e95b2
2019-10-01 00:43:06 UTC	view	5802490	1487580009471148064		shary	5.08	554342223	0b974342-1a53-41c1-a426-23130e770f4b
2019-10-01 00:44:00 UTC	cart	5643943	1487580010100293687			1.05	554342223	0b974342-1a53-41c1-a426-23130e770f4b
2019-10-01 00:44:31 UTC	cart	5809921	1487580010100293687		grattol	2.06	554342223	0b974342-1a53-41c1-a426-23130e770f4b
2019-10-01 00:45:03 UTC	cart	5875322	1487580010100293687			0.92	554342223	0b974342-1a53-41c1-a426-23130e770f4b
2019-10-01 00:56:43 UTC	view	5756762	1487580008909111303			3.02	446419295	d5e0977b-b6ac-43a1-b173-63e9742e95b2
2019-10-01 00:57:25 UTC	view	5697103	1487580008657454075			2.22	544418344	d5e0977b-b6ac-43a1-b173-63e9742e95b2
2019-10-01 03:18:29 UTC	remove_from_cart	5830552	1487580005671109489		masura	1.73	544418344	513b523a-be03-4bcb-85a1-ee0340170a8
2019-10-01 03:18:35 UTC	remove_from_cart	5796770	1487580004916134735			4.29	544418344	513b523a-be03-4bcb-85a1-ee0340170a8
2019-10-01 03:32:00 UTC	view	5653177	1487580012994363565			9.84	429179215	125c71ea-435a-4ef3-9df8-bdcfad1c46a3
2019-10-01 03:37:12 UTC	purchase	5864600	1487580013011140782		italwax	1.24	429179215	125c71ea-435a-4ef3-9df8-bdcfad1c46a3
2019-10-01 03:37:12 UTC	purchase	6731	1487580007256556476			14.27	429179215	125c71ea-435a-4ef3-9df8-bdcfad1c46a3
2019-10-01 03:37:12 UTC	purchase	5770380	1487580012994363565		italwax	10.32	429179215	125c71ea-435a-4ef3-9df8-bdcfad1c46a3

Cuadro 1: Conjunto de eventos de referencia, sobre el que se va montar un índice sobre userID, como clave de búsqueda.

Si extrayésemos información del conjunto de datos completo, sin ningún tipo de estructura de índice, tendríamos que visitar las 20 filas para contar el número de eventos en los que interviene nuestro usuario. En este caso, dicho usuario participa en 5 eventos, y la cantidad total de dinero es $2,86 + 2,86 + 2,7 + 2,86 + 3,02 = 14,30$. Si construyésemos el índice a priori, el funcionamiento sería el siguiente. Comenzamos construyendo dicho índice por usuarios, que es el campo en el que estamos interesados:

- Comenzamos con el primer elemento de la tabla. Como el índice está vacío metemos el elemento al principio, y nos queda el siguiente índice:

Key	Position
55434223	1

- Continuamos recorriendo la tabla, como los primeros eventos pertenecen todos al mismo usuario, todos van entrando consecutivamente, hasta llegar al siguiente índice:

Key	Position
554342223	1
554342223	2
554342223	3
554342223	4

- Llegamos a un usuario nuevo, que al entrar en orden en nuestro índice, se coloca antes que el usuario que traíamos. De nuevo tenemos varios registros seguidos para el mismo usuario, que en este ejemplo metemos simultáneamente, hasta llegar al siguiente índice:



Key	Position
446419295	5
446419295	6
446419295	7
446419295	8
554342223	1
554342223	2
554342223	3
554342223	4

- Ahora, nos volvemos a encontrar con el usuario 554342223. Lo introducimos a continuación de los registros existentes para ese usuario, quedando así el índice:

Key	Position
446419295	5
446419295	6
446419295	7
446419295	8
554342223	1
554342223	2
554342223	3
554342223	4
554342223	9
554342223	10
554342223	11
554342223	12

- De nuevo encontramos registros del usuario 446419295. Los introducimos a continuación de los registros ya existentes para ese usuario, quedando así el índice:

Key	Position
446419295	5
446419295	6
446419295	7
446419295	8
446419295	13
554342223	1
554342223	2
554342223	3
554342223	4
554342223	9
554342223	10
554342223	11
554342223	12

- Nos encontramos a un usuario nuevo, cuyo ID es mayor que el



446419295 pero menor que el 554342223, por lo que se colocará entre los dos, quedando así el índice:

Key	Position
446419295	5
446419295	6
446419295	7
446419295	8
446419295	13
544418344	14
544418344	15
544418344	16
554342223	1
554342223	2
554342223	3
554342223	4
554342223	9
554342223	10
554342223	11
554342223	12

- Y finalmente, el último usuario, cuyo ID es menor que todos los demás, por lo que se coloca al principio:

Key	Position
429179215	17
429179215	18
429179215	19
429179215	20
446419295	5
446419295	6
446419295	7
446419295	8
446419295	13
544418344	14
544418344	15
544418344	16
554342223	1
554342223	2
554342223	3
554342223	4
554342223	9
554342223	10
554342223	11
554342223	12

Cuadro 2: Índice sobre userID, final obtenido a partir del conjunto de Eventos

Una vez construido el índice, podemos hacer búsquedas sobre nuestros usuarios de forma mucho más eficiente. El orden del índice nos aporta la ventaja de poder utilizar el algoritmo de búsqueda binaria. Este algoritmo funciona de la siguiente manera:

1. Tenemos una colección `array` en el que queremos localizar un elemento `elem`
2. Accedemos a la posición intermedia de `array` (`index=size/2`), y la comparamos con `elem`:
 - a) Si `array[index] == elem`, hemos encontrado el elemento y lo devolvemos
 - b) Si `array[index] > elem`, el elemento se tiene que encontrar en la mitad derecha del `array` (si estuviese en la colección)
 - c) Si `array[index] < elem`, el elemento se tiene que encontrar en la mitad izquierda del `array` (si estuviese en la colección)
3. En función del resultado anterior, devolvemos `index`, o repetimos la búsqueda con la mitad correspondiente (utilizando el vector desde el 0 hasta `index`, o desde `index` hasta el final)

Basadas en este algoritmo, dispondremos de dos funciones, que se llamarán `lower_bound` y `upper_bound`. `Lower bound` nos devolverá la primera posición en la que ocurre el elemento que estamos buscando, y `upper bound` nos devolverá la primera posición en la que ocurre el elemento siguiente, para poder obtener todas las entradas del índice que corresponden al valor con el que estamos trabajando.

Aplicamos nuestra función `lower bound` para encontrar la primera ocurrencia del individuo 446419295. Comenzamos con la búsqueda binaria sobre el índice que hemos construido:

- Accedemos a la posición intermedia (posición 10 del índice). Nos encontramos con la pareja [544418344, 14]. Como el ID que vamos buscando es menor que el que hemos encontrado, continuamos la búsqueda en la mitad inferior del vector.
- Accedemos a la posición intermedia de la mitad inferior (posición 5). Encontramos la pareja [446419295, 5]. que contiene al usuario que buscamos, pero aún no sabemos si es la primera ocurrencia. Comenzamos la búsqueda hacia atrás hasta encontrar un usuario distinto.
- Accedemos a la posición 4, en la que encontramos a la pareja [429179215, 20]. Como el id ha cambiado, sabemos que en la posición 5 del índice está la primera ocurrencia del usuario que nos interesa.

En total, hemos necesitado consultar 3 posiciones del índice para localizar el `lower bound`.

Aplicamos ahora la función `upper bound`. Comenzamos la búsqueda binaria:

- Accedemos a la posición intermedia (posición 10 del índice). Nos encontramos con la pareja [544418344, 14]. Como el ID que vamos buscando es menor que el que hemos encontrado, continuamos la búsqueda en la mitad inferior del vector.
- Accedemos a la posición intermedia de la mitad inferior (posición 5). Encontramos la pareja [446419295, 5]. que contiene al usuario que buscamos, pero aún no sabemos si es la primera ocurrencia. Comenzamos la búsqueda hacia delante hasta encontrar un usuario distinto.
- Accedemos a la posición 6, en la que encontramos a la pareja [446419295, 6]. Pertenece al mismo usuario, continuamos.
- Accedemos a la posición 7, en la que encontramos a la pareja [446419295, 7]. Pertenece al mismo usuario, continuamos.
- Accedemos a la posición 8, en la que encontramos a la pareja [446419295, 8]. Pertenece al mismo usuario, continuamos.
- Accedemos a la posición 9, en la que encontramos a la pareja [446419295, 13]. Pertenece al mismo usuario, continuamos.
- Accedemos a la posición 10, en la que encontramos a la pareja [544419344, 14]. Pertenece a un usuario distinto, por lo que hemos encontrado el upper bound.

De esta forma, hemos localizado la primera y la última ocurrencia del usuario 446419295 en nuestro conjunto de eventos (que sabemos que están entre nuestras posiciones 5 y 10 del índice). Ahora, el total de eventos en los que está involucrado el usuario coincide con la diferencia entre el upper bound y el lower bound ($10 - 5 = 5$), y para calcular el precio total sólo tenemos que recorrer los elementos del índice entre el 5 y el 10 acceder a la posición del conjunto de eventos que se nos indica (posiciones 5, 6, 7, 8 y 13) del vector de eventos, extraer el precio de cada uno de ellos, y hacer la suma.

La eficiencia de un método frente al otro es fácilmente observable. Mientras que trabajando con el conjunto de eventos completo hemos tenido que recorrer sus 20 filas para asegurarnos de que encontrábamos todos los eventos en los que participaba nuestro usuario, utilizando el índice sólo hemos tenido que acceder a 10 posiciones del índice (3 cuando buscamos el lower bound y 7 cuando buscamos el upper bound), y a 5 posiciones del conjunto de eventos (las cinco posiciones que hemos obtenido en el índice para nuestro usuario). Estas diferencias serán mucho más notables cuando tengamos conjuntos de cientos o miles de elementos, ya que sólo construiremos el índice una vez, pero haremos varias consultas sobre él.