Práctica 2: Diseño y puesta a punto de programas elementales escritos en C++

2.1. Objetivos de la práctica

El objetivo de esta práctica es doble.

- En la primera parte de la sesión de laboratorio correspondiente a esta práctica cada alumno va a aprender a utilizar la herramienta de depuración GDB (o GNU Debugger) integrada en el entorno Code::Blocks. Un depurador es una herramienta de gran utilidad ya que ayuda al programador a identificar las causas de los errores de sus diseños.
- En la segunda parte de la sesión cada alumno debe diseñar e implementar seis programas C++ que resuelven sencillos programas que, en esencia, se limitan a transformar la información suministrada por el usuario modificando su presentación.

  Cada alumno debe haber iniciado el desarrollo de estos programas antes de acudir a la sesión de prácticas. El código de cada uno de ellos lo puede traer escrito en papel, almacenado en un lápiz de memoria USB, enviándose a sí mismo un correo electrónico con el código o accediendo directamente a `hendrix` a través de un programa como WinSCP o Cyberduck (ver más detalles en la página web de la asignatura).

  En esta segunda parte de la sesión cada alumno podrá consultar al profesor las dudas que le hayan surgido y el profesor podrá supervisar y hacer un seguimiento del trabajo realizado por cada alumno.

  Los seis programas planteados en esta práctica deberían estar desarrollados y disponibles en la cuenta de `hendrix` de cada alumno antes de finalizar la semana en la que cada alumno tiene su sesión de prácticas.

2.2. Tecnología y herramientas

Los programas con los que se va a trabajar en esta práctica se asociarán a una nueva área de trabajo denominada «Práctica 2», que se ubicará en la carpeta `Practica2`, creada previamente dentro de la carpeta `Programacion1`.

2.2.1. Construcción de programas ejecutables en modo *«Release»* y *«Debug»*

Se creará en el área de trabajo «Práctica 2» un primer proyecto denominado «`TablasMultiplicar`» en el cual se desarrollará el programa cuyo código ha sido presentado en el capítulo 4 del texto de

la asignatura. Dicho programa, que se editará en un fichero denominado `tablas.cc` o `tablas.cpp`, presenta por pantalla las tablas de multiplicar que selecciona interactivamente el usuario.

Al código de este programa se puede acceder desde la sección de «Material docente» de la web de la asignatura a través del enlace «Código C++ y datos».

Si un programa está libre de errores podemos compilarlo y construir un programa ejecutable o aplicación listo para su explotación desde dentro o desde fuera del entorno Code::Blocks. Para ello, debe estar seleccionado previamente el modo *«Release»* ejecutando, en su caso, la orden *«Build»* → *«Select target»* → *«Release»*.

En cambio, si lo que se pretende es construir un programa ejecutable que permita depurar el programa desde el propio Code::Blocks (es decir, ejecutar el programa paso a paso para detectar y corregir errores), antes de proceder a compilarlo y construir un programa ejecutable hay que seleccionar el modo *«Debug»* ejecutando, en su caso, la orden *«Build»* → *«Select target»* → *«Debug»*.

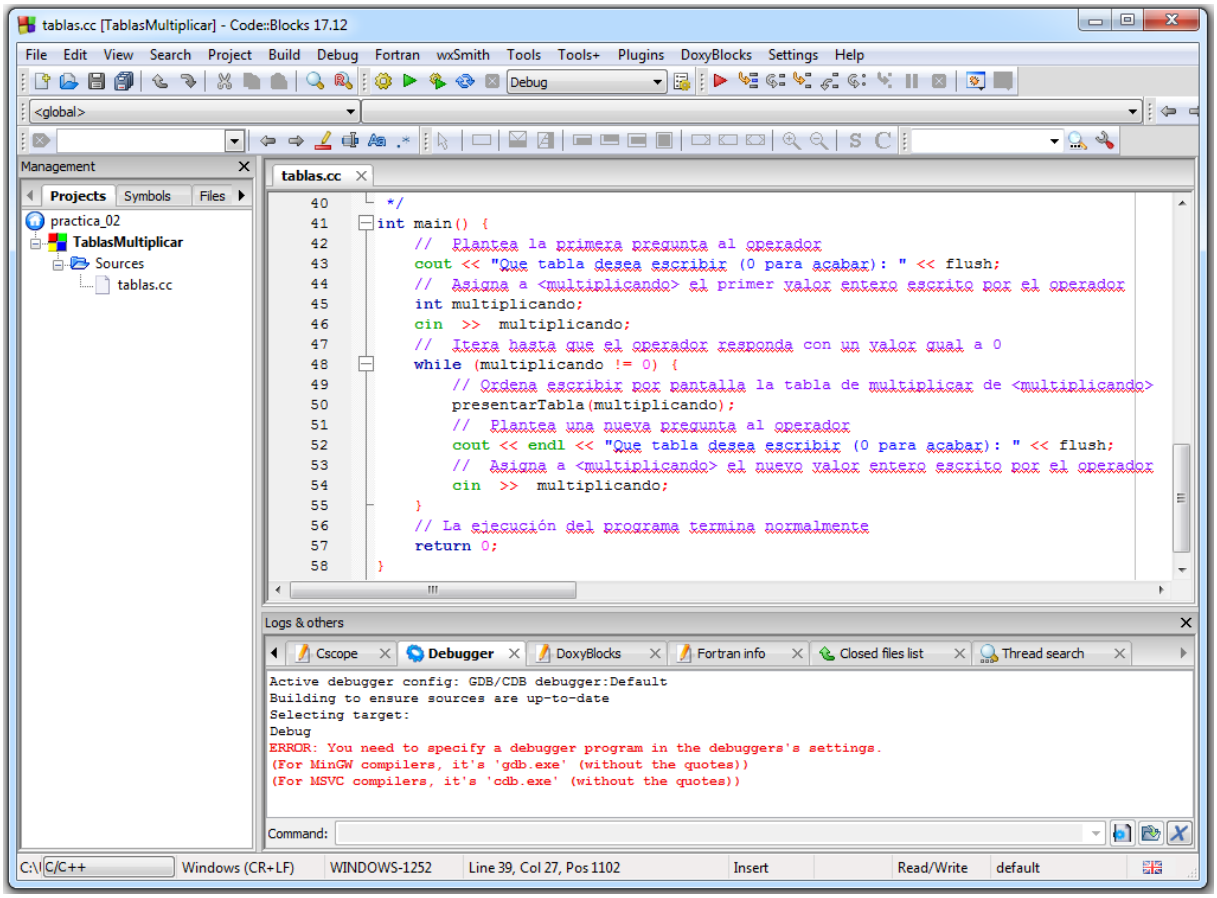
Cada alumno debe compilar el proyecto «`TablasMultiplicar`» en modo *«Release»* y en modo *«Debug»*, debe construir los correspondientes programas ejecutables, debe observar los ficheros generados en cada caso y debe ejecutar ambos programas desde dentro del propio Code::Blocks y desde fuera (invocando directamente la ejecución del programa).

2.2.2. Configuración y uso del depurador GNU Debugger

Configuración de GNU Debugger

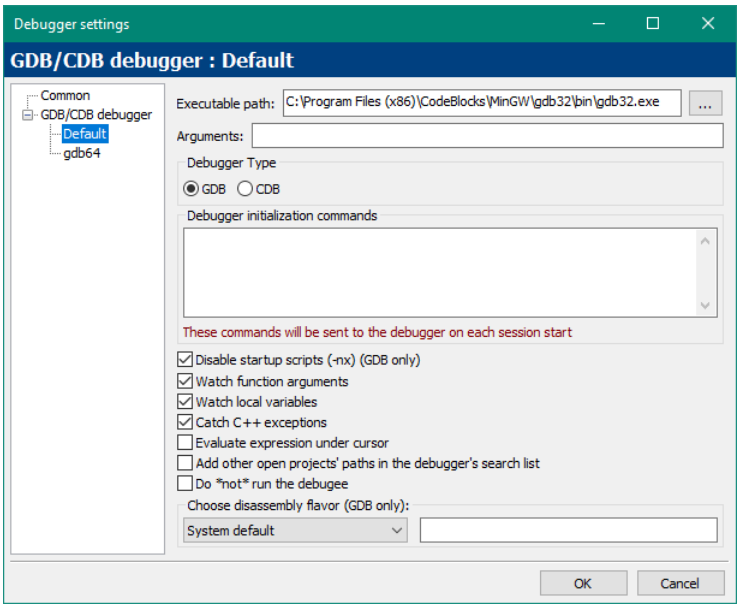
Para hacer uso del depurador GNU Debugger (GDB) desde Code::Blocks es necesario que el depurador conozca la ubicación de dicho programa en nuestro sistema informático.

Tras instalar Code::Blocks es posible que dicha ubicación no haya sido correctamente definida. En tal caso, un mensaje de error advierte que el depurador (*debugger* en inglés) no ha sido especificado y sugiere hacerlo.



El problema se resuelve ejecutando la orden *«Settings»* → *«Debugger...»* → *«Default»* y definiendo la ruta absoluta de acceso al fichero ejecutable correspondiente a GDB (`gdb32.exe` en sistemas operativos Windows, o `gdb` en otros sistemas) en el campo de texto *«Executable path:»*. Suele estar almacenado en misma carpeta donde se almacena el compilador de C++ utilizado. En el caso de Windows, si se descargó el instalador de Code::Blocks que incluía el compilador MinGW y se han mantenido los directorios de instalación predeterminados, la ruta será `C:\Program Files (x86)\CodeBlocks\MinGW\gdb32\bin\gdb32.exe` o `C:\Program Files\CodeBlocks\MinGW\gdb32\bin\gdb32.exe`. Si se ha instalado con otro instalador o en otro sistema operativo, puede averiguarse la ruta del compilador por defecto en Code::Blocks[1] y buscarse en ella o en sus subdirectorios el fichero `gdb32.exe` o `gdb` y configurarse.

[1] *«Settings»* → *«Compiler...»* → *«Global compiler settings»* → *«Selected compiler»* → *«GNU GCC Compiler»* → *«Toolchain executables»* → *«Compiler's installation directory»*.



Utilización del depurador GDB

Estamos en condiciones de usar el depurador GDB para observar el comportamiento del programa desarrollado en el proyecto «`TablasMultiplicar`».

Para ello vamos a definir un *punto de interrupción* (en inglés, *breakpoint*) en el código del programa. Ello se logra seleccionando con el ratón la línea dónde se desea definir el punto de interrupción y haciendo clic en el margen de la misma, a la derecha del número de línea. Ello nos permitirá ejecutar el programa a través de la opción de menú *«Debug»* → *«Start / Continue»* hasta que el flujo del programa alcance ese punto de interrupción y poder observar el valor de los datos en ámbito en ese punto.

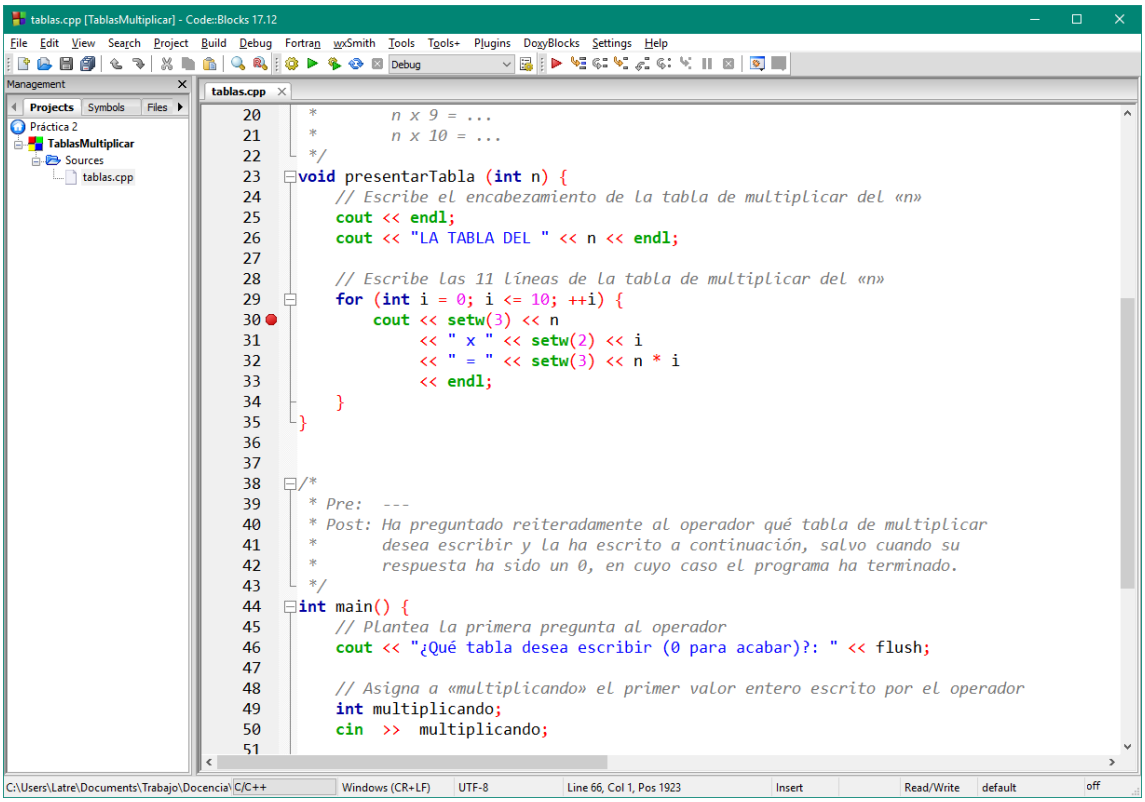
En un programa podemos definir cuantos puntos de interrupción nos interese. Para eliminar un punto de interrupción, basta con hacer clic sobre él.

Para depurar el proyecto «`TablasMultiplicar`» se sugiere seguir el siguiente guion:

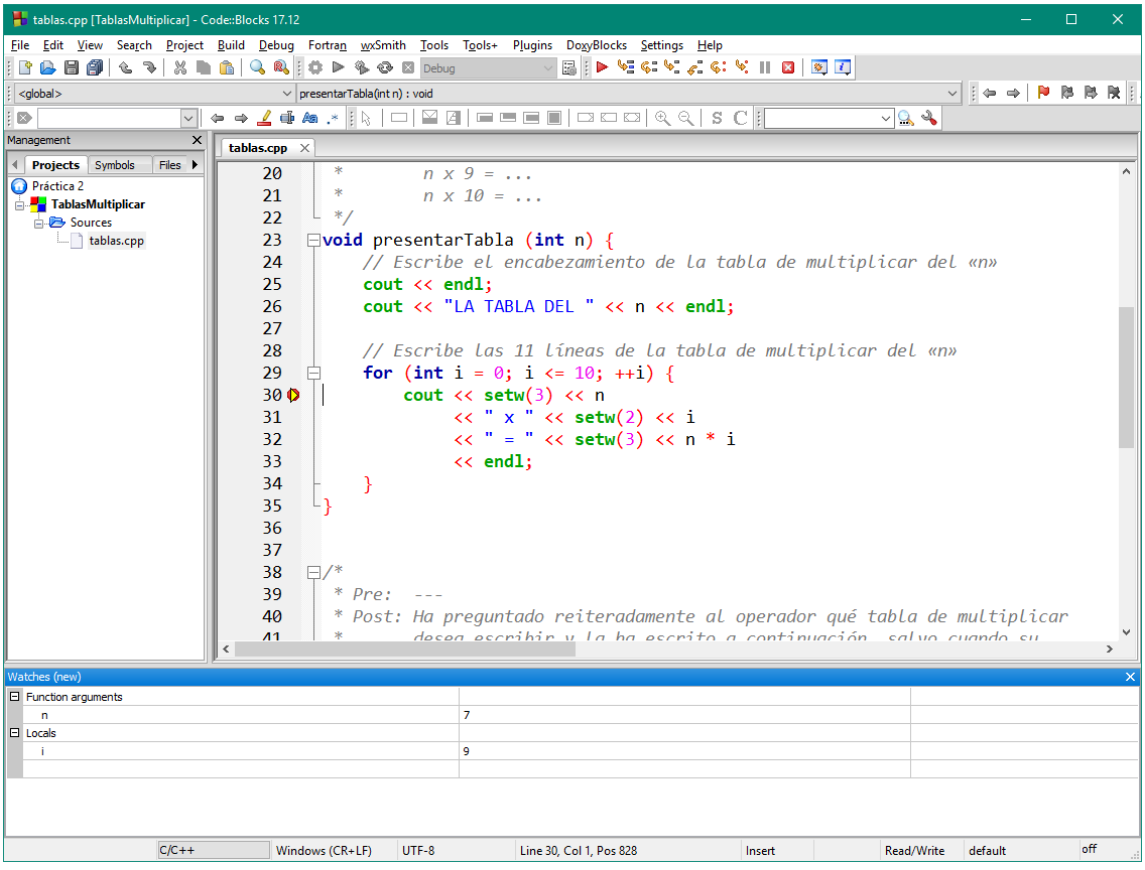
1. Consultar y verificar que el modo *«Debug»* está seleccionado (opción de menú *«Build»* → *«Select target»* o comprobando que el modo *«Debug»* aparece seleccionado en la barra de herramientas *«Compiler»*). Si no lo estuviera, proceder a seleccionarlo tal como se ha explicado anteriormente.

2. Construir un programa ejecutable activando la orden *«Build»* → *«Build»*.

3. Definir un punto de interrupción en la línea correspondiente a la única instrucción a iterar dentro del bucle **`for`** programado en la función `presentarTabla(n)` (línea 30 del código). El punto de interrupción queda identificado por un punto rojo como se aprecia en la siguiente figura:



4. Abrir una ventana *«Watches»* que permita observar el valor de las variables en ámbito en cada punto de ejecución del programa. Si no está abierta, se puede abrir ejecutando la orden *«Debug»* → *«Debugging windows»* → *«Watches»*.



5. Ejecutar el programa hasta el siguiente punto de interrupción mediante la orden *«Degub»* → *«Start / Continue»* o pulsando el icono equivalente (un triángulo rojo con vértice a la derecha). Esta orden permite avanzar en la ejecución del programa hasta el siguiente punto de interrupción (o hasta el final del programa) y observar la evolución de los datos visibles en cada punto de interrupción.

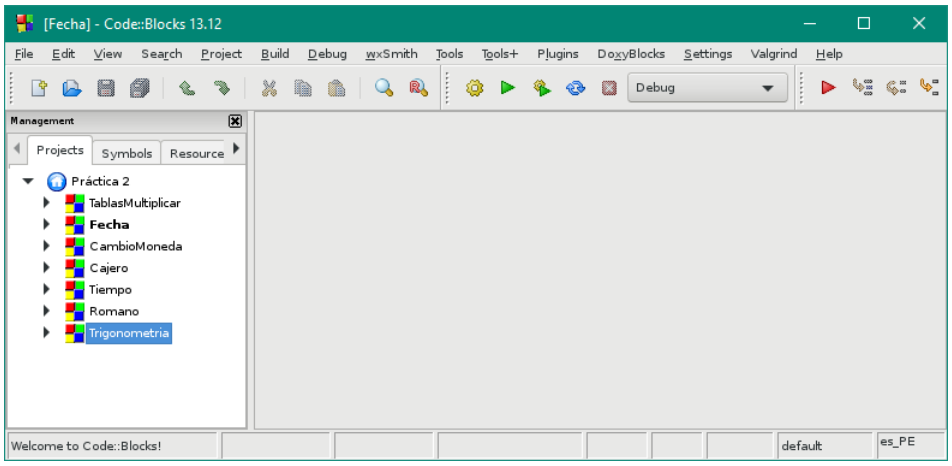
Se recomienda estudiar las posibilidades del depurador GBD explorando las opciones disponibles en los diferentes menús desplegables y el significado de los botones que aparecen en la barra de herramientas.

2.3. Trabajo a desarrollar en esta práctica

Los problemas que aquí se proponen requieren transformar la información proporcionada por el usuario del programa con un determinado formato en una información equivalente con diferente formato. La información tratada en cada problema es de diferente naturaleza: una fecha del calendario, una cantidad de dinero expresada en una determinada moneda (euro, peseta, etc.), la medida de un ángulo, la duración de un periodo de tiempo, etc.

Cada problema exige la puesta a punto de un programa completo en el cual se leen los datos que proporciona el usuario y se calculan y presentan por pantalla los resultados que correspondan.

Los programas que resuelven estos seis problemas se desarrollarán en el área de trabajo «Práctica 2». En ella se definirán seis nuevos proyectos, uno por cada uno de los programas a desarrollar: «`Fecha`», «`CambioMoneda`», «`Cajero`», «`Tiempo`», «`Romano`» y «`Trigonometria`». La descripción del comportamiento de los programas a desarrollar se detallan en las secciones 2.3.1 a 2.3.6.



Este trabajo de desarrollo de programas debe ir acompañado por la realización de un conjunto de pruebas en búsqueda de posibles errores. Esta estrategia se aplicará a todos los trabajos de programación que se lleven a cabo en esta asignatura, ya que el desarrollo de un programa no concluye hasta que se han realizado todas las pruebas necesarias para validar su buen comportamiento. El objeto de estas pruebas es localizar errores y fallos de funcionamiento para corregir, en su caso, sus causas en el código fuente y lograr el programa se comporte respetando sus especificaciones para cualquier juego de datos.

En las descripciones de los programas que han de desarrollarse, se dan algunas pautas para la realización de pruebas sobre cada uno de ellos. En todos los casos, se propone reflexionar sobre los datos con los que probarlos y sobre los resultados que debería ofrecer el programa antes incluso de haberlo programado, ya que dicha reflexión previa puede ayudar a terminar de comprender el problema. Es

por ello aconsejable determinar *a mano*, antes de comenzar el desarrollo los programas, los resultados esperados de los casos de prueba que se establezcan. Una vez desarrollado el programa, será necesario comparar los resultados ofrecidos por el programa con los resultados esperados al diseñar el caso de prueba, y corregir los errores que pueda haber.

En una primera etapa, desarrolla los seis programas propuestos suponiendo que, cuando al usuario se le solicitan datos numéricos que cumplen con una serie de características (por ejemplo, ser positivo), este los va a proporcionar cumpliendo dichas características. Los casos de prueba, por lo tanto, se centrarán en verificar que el programa funciona correctamente con entradas de usuario válidas.

En una segunda etapa (ver sección 2.3.7), se te pedirá modificar ligeramente los programas realizados para que estos sean algo más robustos ante entradas del usuario que no cumplan con todas las restricciones que se le impongan.

2.3.1. Proyecto «`Fecha`». Programa de cambio de formato de una fecha

Debe desarrollarse un programa C++ que interaccione con el usuario mostrando el siguiente comportamiento:

```
Escriba una fecha con formato de 8 cifras [aaaammdd]: 14921012
La fecha escrita es 12/10/1492.
```

La fecha escrita por el usuario con formato *aaaammdd*, que se ha subrayado para mayor claridad, es reescrita por el programa con formato *dd/mm/aaaa*, donde *dd* es el día de la fecha, *mm* es el mes y *aaaa* es el año. **El usuario siempre va a escribir números positivos que representan fechas válidas con formato *aaaammdd*** (el mes estará comprendindo entre 1 y 12; el día estará entre 1 y el último día del mes correspondiente).

Un segundo ejemplo se presenta a continuación para ilustrar el comportamiento deseado del programa. Obsérvese que los días y meses se escriben con dos dígitos, donde el primero de ellos será un 0 si el número de día o de mes es inferior a 10.

```
Escriba una fecha con formato de 8 cifras [aaaammdd]: 20140706
La fecha escrita es 06/07/2014.
```

Fíjate como en este proyecto «`Fecha`» se han proporcionado dos ejemplos distintos de ejecución que cubren la posibilidad de que tanto el día como el mes tengan una o dos cifras.

2.3.2. Proyecto «`CambioMoneda`». Programa de conversión de una moneda a otra

El programa interactivo que debe ser desarrollado pide al usuario que escriba una cantidad de dinero **no negativa** expresada en euros y procede a informarle de su desglose en euros y céntimos y también de su equivalente en pesetas.

```
Escriba una cantidad positiva de dinero en euros: 43.653
Son 43 euros y 65 céntimos que equivalen a 7263 pesetas.
```

Obsérvese que el programa redondea la cantidad de céntimos al céntimo más próximo y que la cantidad de pesetas equivalentes es también redondeada al número entero de pesetas más próximo al resultado exacto.

```
Escriba una cantidad positiva de dinero en euros: 43.6583
Son 43 euros y 66 céntimos que equivalen a 7264 pesetas.
```

Este proyecto, además de ser probado con los ejemplos del enunciado, conviene ser probado con los valores límite mínimo y máximo que provocan que cada uno de los tres datos de salida (euros enteros, céntimos y pesetas) pasen de 0 a 1. Dado que la entrada del usuario es de tipo real, vamos a tratar de obtener esos valores con tres decimales como máximo, completando la tabla que aparece a continuación, donde cada fila representa una prueba específica a partir de una entrada de usuario concreta. Algunas pruebas ya están completas, mientras que en otras tendrás que completar la entrada de usuario, los resultados esperados o ambos. Cálculos, a mano o con una calculadora, antes de diseñar el programa. Una vez desarrollado, comprueba cuál es el comportamiento del programa y si difiere de las pruebas diseñadas, corrige los errores detectados.

		Resultado esperado		
Entrada usuario	**Razón del caso de prueba**	**Euros**	**Céntimos**	**Pesetas**
43,653	Primer ejemplo del enunciado	43	65	7263
43,6583	Segundo ejemplo del enunciado	43	66	7264
0	Mínimo valor posible de la entrada del usuario			
0,001	Mínimo valor posible no nulo de la entrada del usuario (considerando solo 3 decimales)			
	Máximo valor cuya conversión es 0 pesetas			0
	Mínimo valor cuya conversión es 1 peseta y máximo que corresponde con 0 céntimos		0	1
	Mínimo valor cuyo redondeo corresponde con 1 céntimo		1	
	Máximo valor que corresponde con 0 euros enteros	0		
	Mínimo valor que corresponde con 1 euro entero	1		
	Un valor entero (sin decimales) distinto de 0			

2.3.3. Proyecto «`Cajero`». Programa sobre el funcionamiento de un cajero automático

El programa interactivo a desarrollar presenta el siguiente comportamiento:

```
Cantidad a retirar en euros [positiva y múltiplo de 10]: 280

Billetes   Euros
========   =====
    1        10
    1        20
    5        50
```

El programa informa al usuario del número de billetes que le devolverá un cajero al retirar la cantidad de dinero por él especificada, que será **positiva y múltiplo de 10**. Conviene advertir que el cajero

dispone únicamente de billetes de diez, de veinte y de cincuenta euros y que siempre minimizará el número de billetes a entregar.

```
Cantidad a retirar en euros [positiva y múltiplo de 10]: 590

Billetes   Euros
========   =====
    0        10
    2        20
   11        50
```

Además de con los ejemplos anteriores, haz pruebas con los valores mínimo y máximo que se pueden sacar de un cajero automático: *10* y *600* euros, respectivamente. Haz pruebas también con las tres cantidades que obligan al cajero a entregar exactamente un billete, ya sea de 10, 20 o 50 euros. Haz también una prueba adicional con la cantidad de dinero que obliga al cajero a entregar un total de tres billetes, siendo exactamente uno de cada tipo.

Puedes organizar el diseño de estas pruebas en una tabla con la del proyecto anterior. Igualmente, es recomendable realizar el diseño de estas pruebas antes que el del programa.

2.3.4. Proyecto «`Tiempo`». Programa sobre la medida del tiempo

El programa interactivo a desarrollar pide al usuario que exprese el tiempo de duración de un determinado evento, como una cantidad **natural** expresada en segundos, para informar a continuación por la pantalla de la equivalencia del tiempo introducido en días, horas, minutos y segundos.

```
Duración en segundos: 615243
Este tiempo equivale a 7 días 2 horas 54 minutos y 3 segundos
```

Un segundo ejemplo que muestra los resultados que presenta el programa.

```
Duración en segundos: 11412
Este tiempo equivale a 0 días 3 horas 10 minutos y 12 segundos
```

Haz pruebas, ademas de con estos dos ejemplos de ejecución, con el mínimo valor posible de la duración (0 segundos), con las tres duraciones máximas que dan equivalencias estrictamente inferiores a un minuto, a una hora y a un día, así como con las cuatro duraciones mínimas que equivalen exactamente a un segundo, un minuto, una hora y un día. Como en los casos anteriores, organiza los casos de prueba en una tabla y diséñalos antes que el programa.

2.3.5. Proyecto «`Romano`». Programa sobre escritura de números romanos

El programa interactivo a desarrollar presenta el siguiente comportamiento:

```
Escriba un entero entre 1 y 10: 8
8 = VIII
```

El programa pregunta al usuario por un número entero del intervalo [1, 10] y le informa por pantalla de su equivalencia como número romano.

```
Escriba un entero entre 1 y 10: 5
5 = V
```

Finalmente se muestra el diálogo entre programa y usuario correspondiente a una nueva ejecución del programa.

```
Escriba un entero entre 1 y 10: 4
4 = IV
```

El proyecto «`Romano`» debes probarlo con todos los valores comprendidos entre 1 y 10, ambos inclusive.

2.3.6. Proyecto «`Trigonometria`». Programa sobre ángulos y funciones trigonométricas

El programa interactivo a desarrollar pide al usuario que defina un ángulo en grados, minutos y segundos sexagesimales y le informa por pantalla, a continuación, del valor equivalente de ese ángulo en radianes, así como de los valores de su seno, coseno y tangente. El programa desarrollado debe presentar los resultado respetando al máximo el formato mostrado en los siguientes ejemplos (los ángulos en radianes se expresan con tres cifras decimales y los valores de las funciones trigonométricas seno, coseno y tangente, con cuatro decimales).

```
Escriba el valor de un ángulo (grados, minutos y segundos): 60   0 0
Valor del ángulo en radianes: 1.047 radianes
sen 1.047 = 0.8660
cos 1.047 = 0.5000
tg 1.047 = 1.7321
```

Un segundo ejemplo que muestra los resultados que presenta el programa:

```
Escriba el valor de un ángulo (grados, minutos y segundos): 112 9    45
Valor del ángulo en radianes: 1.958 radianes
sen 1.958 = 0.9261
cos 1.958 = -0.3772
tg 1.958 = -2.4550
```

Este proyecto, además de ser probado con los ejemplos del enunciado, conviene ser probado con ejemplos en los que tanto los grados, como los minutos y los segundos tomen los valores que razonablemente se considerarían como límites mínimo y máximo razonables (0 en cuanto al mínimo de los tres, y 359 o 59 en lo que respecta a los máximo para grados por un lado y minutos y segundos, por otro). También se debería probar con la menor entrada posible no nula (0º 0’ 1”) y con una entrada que sea mayor o igual que 360 grados. También nos podemos plantear que debería ocurrir en el caso de

entradas más extrañas, como por ejemplo, que en algún caso los minutos o los segundos son mayores o iguales que 60 (30º 90’ 0” no es la forma más elegante de escribir el ángulo 31º 30’ 0”, pero no es ningún error).

Completa para ello la siguiente tabla:

Entrada usuario				**Resultado esperado**			
º	**’**	**”**	**Razón del caso de prueba**	**rad**	**sen**	**cos**	**tg**
60	0	0	Primer ejemplo del enunciado	1,047	0,8660	0,5000	1,7321
112	9	45	Segundo ejemplo del enunciado	1,958	0,9261	-0,3772	-2,4550
0	0	0	Mínimo valor posible de la entrada del usuario				
0	0	1	Mínimo valor no nulo				
359	59	59	Máximos habituales para grados, minutos y segundos				
			Más de 360°				
			Más de 60 minutos				
			Más de 60 segundos				

2.3.7. Robustez de los programas desarrollados

Al desarrollar y probar los programas anteriores, se ha trabajando bajo la hipótesis de que el usuario va a proporcionar entradas válidas cuando le son solicitadas (por ejemplo, si se le pide un entero positivo múltiplo de diez, el usuario va a escribir un dato que cumpla con dichas características). En esta sección, se va a aumentar la robustez de los programas desarrollados, desconfiando (solo ligeramente) de las entradas proporcionadas por el usuario y comprobando que las restricciones que se le hayan impuesto en cada problema realmente se cumplen.

Lamentablemente, va a ser posible hacerlo solo de forma muy limitada, ya que en todos y cada uno de los problemas se está solicitando al usuario datos de tipo numérico, que son leídos del teclado directamente a variables de tipo entero o real. Las técnicas para detectar y resolver el problema de que el usuario introduzca información no numérica por el teclado requerirían cambios profundos en los programas y no forman parte del objetivo de este curso. En cambio, sí que va a ser posible comprobar que los datos numéricos facilitados cumplan con las restricciones que se han impuesto en cada problema:

Proyecto «`Fecha`». Pese a que el enunciado asegura que el usuario va a introducir siempre datos que representan fechas correctas, piensa (antes de ejecutarlo) en cómo debería comportarse el programa ante:

1. Un entero negativo como -20181012.
2. Un entero que no represente una fecha del calendario (por codificar un día del mes que no esté entre 1 y 31 o un mes que no esté entre 1 y 12), como, por ejemplo, 20181033, 20191000, 20210015 o 20221315.
3. Un dato de tipo real como 20190225.75 (que un usuario podría estar pensando que representa las seis de la tarde del 25 de febrero de 2019).

En los dos primeros casos, quizá no estaría de más que el programa escribiera un mensaje de error en la pantalla en lugar de intentar escribir ninguna fecha y que la función `main` devolviera un entero positivo en lugar de 0, indicando que la ejecución no ha sido correcta. En el tercero, podría considerarse que para escribir la fecha correspondiente a la entrada del usuario solo es necesaria la parte entera. Quizá tu programa ya funcione correctamente de acuerdo con este criterio y no haya que realizar ninguna modificación.

Proyecto «`CambioMoneda`». Añade casos de prueba a la tabla de los casos de prueba de este proyecto en el que introduzcas cantidades negativas. Los ejemplos del enunciado, con signo negativo, pueden valer. Antes de ejecutar el programa de nuevo, completa las celdas de las columnas correspondientes al resultado esperado.

		Resultado esperado		
Entrada usuario	**Razón del caso de prueba**	**Euros**	**Céntimos**	**Pesetas**
-43,653	Dato negativo			
-43,6583	Dato negativo			

Después de haber calculado a mano cuáles deberían ser los resultados esperados, ejecuta el programa. Analiza por qué los resultados del programa no son correctos.

Antes de pensar en ninguna solución complicada que resuelva la discrepancia encontrada, habría que tener en cuenta que el programa explícitamente pide una cantidad positiva de dinero. Se puede aumentar la robustez de este programa simplemente escribiendo un mensaje de error en la pantalla cuando la entrada del usuario sea negativa, y haciendo que la función `main` devuelva un entero positivo en lugar de 0, indicando que la ejecución no ha sido correcta.

Proyecto «`Cajero`». Las clases de entrada numérica de usuario que no cumplirían con la restricción de que fuese un entero positivo y múltiplo de 10 son:

1. Enteros negativos
2. Enteros positivos que no sean múltiplos de 10
3. Datos reales con decimales

En el primer caso, el cajero no puede proporcionar billetes y en el segundo y tercero, no podrían proporcionar toda la cantidad de dinero solicitada, por lo que sería adecuado considerarlos como situaciones de error y limitarse escribir un mensaje de error en la pantalla en lugar de escribir el número de billetes de cada tipo en la pantalla. Desafortunadamente, con lo estudiado hasta ahora en la asignatura, detectar el tercer caso requeriría hacer modificaciones importantes en el programa. Puedes, simplemente, conformarte con escribir mensajes de error en el caso de enteros negativos o no múltiplos de 10.

Proyecto «`Tiempo`». Las clases de entrada de usuario que no cumplirían con la restricción de que fuese un entero positivo serían similares a las del problema anterior:

1. Enteros negativos
2. Datos reales positivos con decimales
3. Datos reales negativos con decimales

En cambio, ahora no está tan claro que se trate de situaciones de error, en particular la primera de ellas. ¿Cuáles deberían ser las respuestas dadas por el programa ante cada una de estas clases de entrada? Contesta a la pregunta antes de ejecutar el programa de nuevo, y si el resultado del programa difiere del que consideras que tendría que tener, corrígelo.

Proyecto «`Romano`». ¿Qué datos enteros no serían entradas válidas? Modifica el programa para que en esos casos, muestre un mensaje de error y la función `main` devuelva un valor positivo indicando un error.

En el caso de este proyecto, los datos reales con decimales serían también entradas no válidas, pero de nuevo, detectar dicha circunstancia puede ser complicado, con lo que puedes ignorar que se puedan producir este tipo de entradas.

Proyecto «`Trigonometria`». En el caso de este proyecto, cabe plantearse que debería ocurrir en el caso de entradas más problemáticas: ¿qué debería ocurrir si se expresa el ángulo con una cantidad negativa de grados? ¿O si solo los minutos o los segundos son negativos? ¿Y si la entrada es de exactamente 90 o 270 grados? Conviene decidir antes de comprobar cómo se comporta el programa si esas situaciones son de error o no (no necesariamente lo son) y, en el caso de no serlo, cuál debe ser el comportamiento del programa.

Entrada usuario				**Resultado esperado**			
º	**'**	**"**	**Razón del caso de prueba**	**rad**	**sen**	**cos**	**tg**
			Grados negativos (minutos y segundos nulos o positivos)				
			Minutos negativos (grados y segundos nulos o positivos)				
			Segundos negativos (grados y minutos nulos o positivos)				
90	0	0	Tangente no definida	1,571	1,0000	0,0000	¿∞?
270	0	0	Tangente no definida	1,571	1,0000	0,0000	¿$-\infty$?

2.3.8. Resultados del trabajo desarrollado en las dos primeras prácticas

Como resultado de las dos primeras prácticas, cada alumno dispondrá en su cuenta `hendrix-ssh` de una carpeta denominada `Programacion1` dentro de la cual se localizarán los siguientes proyectos de programación en C++ organizados en las áreas de trabajo y con las denominaciones que se detallan a continuación:

1. Área de trabajo «Práctica 1», en la carpeta `Programacion1/Practica1`, con los siguientes proyectos ubicados en ella:
   - Proyectos C++ «`Bienvenida`», «`Circunferencias`», «`Circulo`» y «`Formatear`»
2. Área de trabajo «Práctica 2», en la carpeta `Programacion1/Practica2`, con los siguientes proyectos ubicados en ella:
   - Proyecto C++ «`Fecha`»
   - Proyecto C++ «`CambioMoneda`»
   - Proyecto C++ «`Cajero`»
   - Proyecto C++ «`Tiempo`»
   - Proyecto C++ «`Romano`»
   - Proyecto C++ «`Trigonometria`»

   Estos proyectos funcionarán de forma obligatoria de acuerdo con las especificaciones dadas en las secciones 2.3.1 a 2.3.6. Las modificaciones solicitadas en la sección 2.3.7 son voluntarias, aunque muy recomendables.