# 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 acabar de poner a punto seis programas C++ que resuelven sencillos programas que, en esencia, se limitan a transformar la información suministrada por el operador modificando su presentación.

  Cada alumno debe haber desarrollado 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 dela 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 deben 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`, 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»*.

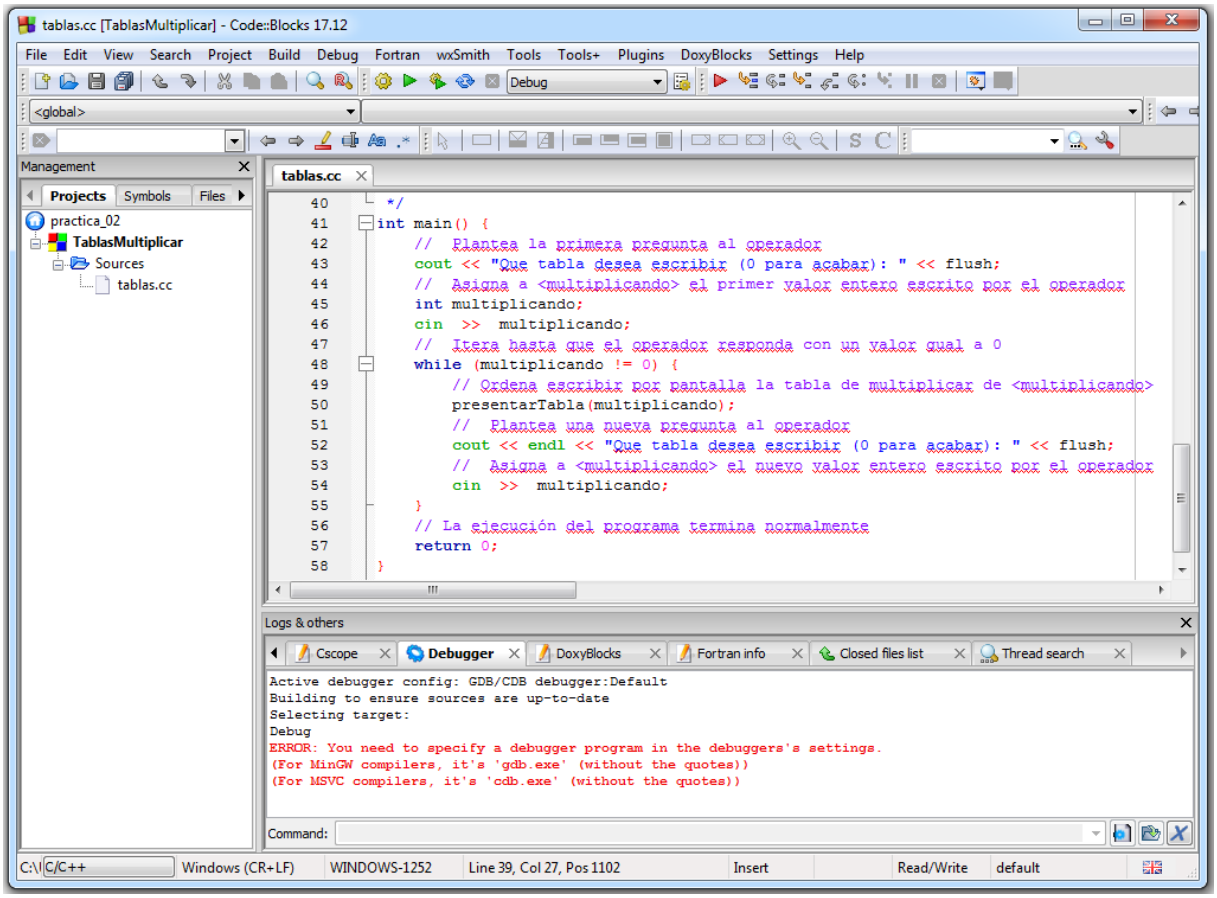
Cada alumno debe compilar el proyecto «`TablasMultiplicar`» en modo *«Release»* y en modo *«Debug»*, debe construir los correspondientes programas ejecutables, debe observar los ficheros generados en cada caso y debe ejecutar ambos programas desde dentro del propio Code::Blocks y desde fuera (invocando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

### 2.2.2. Configuración y uso del depurador GNU Debugger

#### Configuración de GNU Debugger

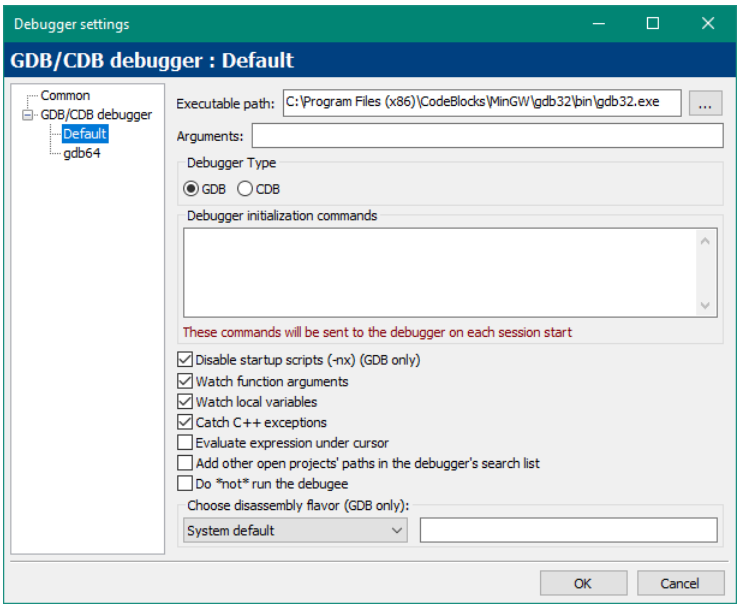
Para hacer uso del depurador GNU Debugger (GDB) desde Code::Blocks es necesario que el depurador conozca la ubicación de dicho programa en nuestro sistema informático.

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 *«Degub»* → *«Start / Continue»* hasta que el flujo del programa alcance ese punto de interrupción interrupción y poder observar el valor de los datos en ámbito en ese punto.

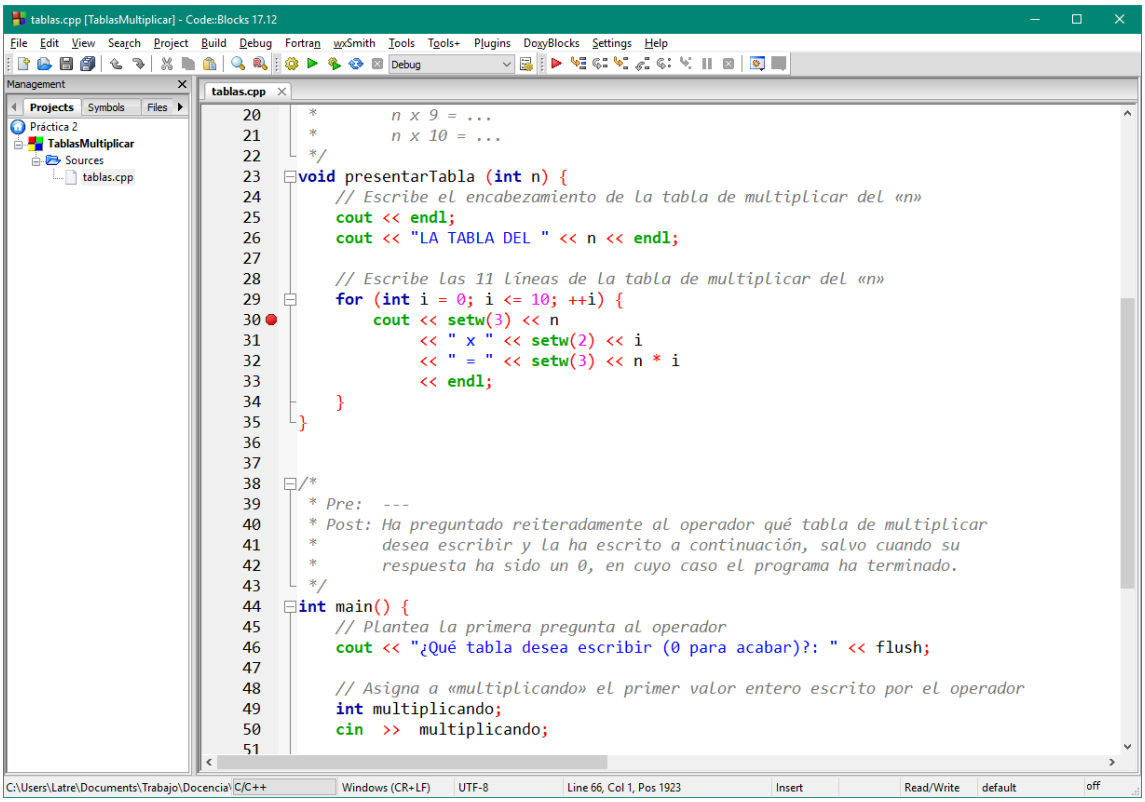
En un programa podemos definir cuantos puntos de interrupción nos interese. Para eliminar un punto de interrupción, basta con hacer clic sobre él.

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

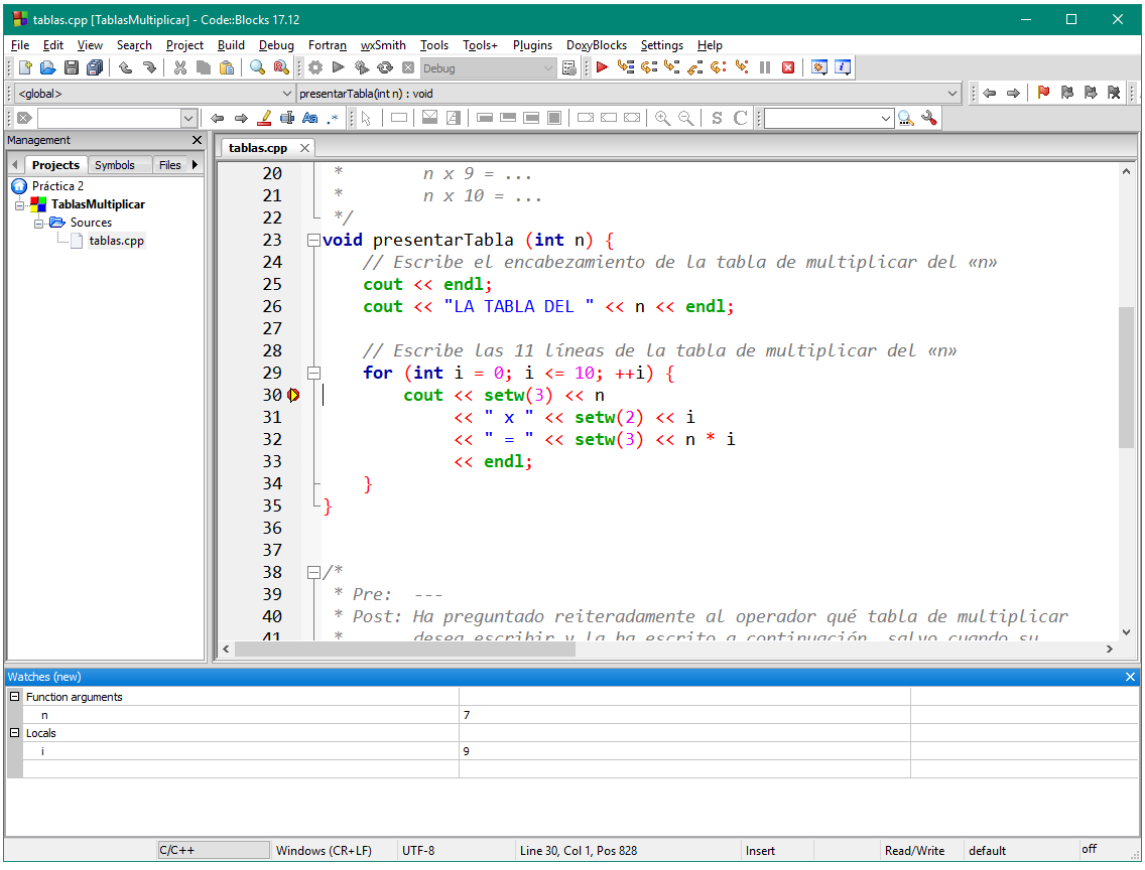
1. Consultar y verificar que el modo *«Debug»* está seleccionado (opción de menú *«Build»* → *«Select target»* o comprobando que el modo *«Debug»* aparece seleccionado en la barra de herramientas *«Compiler»*). Si no lo estuviera, proceder a seleccionarlo tal como se ha explicado anteriormente.

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.

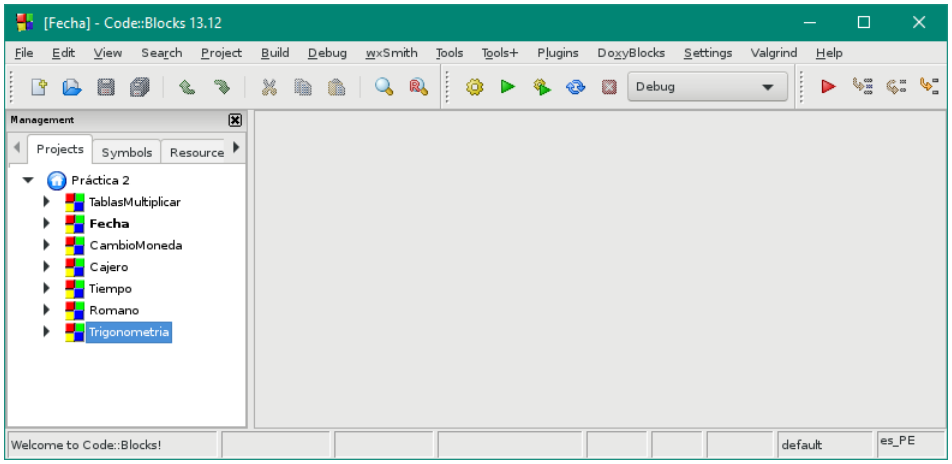
Se recomienda estudiar las posibilidades del depurador GBD explorando las opciones disponibles en los diferentes menús desplegables y el significado de los botones que aparecen en la barra de herramientas.

## 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.), un ángulo de un polígono, 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 operador 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`».




Este trabajo de desarrollo de programas debe ir acompañado por la realización de un conjunto de pruebas en búsqueda de posibles errores, tal como se describe más adelante. Esta estrategia se aplicará a todos los trabajos de programación que se lleven a cabo en esta asignatura.

Los programas a desarrollar, con la descripción de su comportamiento, y el nombre de sus correspondientes proyectos se detallan a continuación.

### 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 operador 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.

Un segundo ejemplo 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.
```

### 2.3.2. Proyecto CambioMoneda. Programa de cambio de moneda

El programa interactivo que debe ser desarrollado pide al operador que escriba una cantidad de dinero expresada en euros y procede a informarle de su desglose en euros y céntimos y también de su equivalente en pesetas.

```
Escriba una cantidad de dinero en euros: 43.653
Son 43 euros y 65 centimos 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 de dinero en euros: 43.6583
Son 43 euros y 66 céntimos que equivalen a 7264 pesetas
```

### 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 operador del número de billetes que le devolverá un cajero al retirar la cantidad de dinero por él especificada. 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]: 1290

Billetes   Euros
========   =====
    0        10
    2        20
   25        50
```

### 2.3.4. Proyecto Tiempo. Programa sobre la medida del tiempo

El programa interactivo a desarrollar pide al operador que exprese el tiempo de duración de un determininado evento, en segundos, para informar por pantalla, a continuación, de la equivalencia de ese tiempo en días, horas, minutos y segundos.

```
Duración en segundos: 615242.83
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
```

El resultado presenta la cantidad de segundos redondeada al segundo más próximo.

```
Duración en segundos: 132.896
Este tiempo equivale a 0 días 0 horas 2 minutos y 13 segundos
```

### 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 operador 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 operador correspondiente a una nueva ejecución del programa.

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

### 2.3.6. Proyecto Trigonometria. Programa sobre ángulos y funciones trigonométricas

El programa interactivo a desarrollar pide al operador 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
```

### 2.3.7. Realización sistemática de pruebas

El desarrollo de un programa no concluye hasta que se han realizado todas las pruebas necesarias para validar su buen comportamiento. Por tanto, los programas desarrollados en esta práctica no pueden considerarse terminados hasta que no se haya realizado una variedad de pruebas del funcionamiento de los mismos suficiente para tener la convicción de 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 los siguientes párrafos se dan algunas pautas para la realización de pruebas sobre cada uno de los programas a desarrollar en esta práctica 2.

Recuerda que el objetivo de hacer pruebas es encontrar defectos en los programas desarrollados, por lo que no deben darse directamente por buenas las respuestas que tus programas den cuando los estés probando con casos triviales. El programa debe funcionar correctamente también en los casos no triviales, en los casos más comprometidos. Para ello debes identificar esos casos, determinar *'a mano'* los resultados esperables, y comparar éstos con los que proporciona el programa al ser ejecutado.

Así, el proyecto **Fecha**, además de probarlo con los datos del primero de los ejemplos del enunciado, conviene probarlo con una fecha cuyo día esté comprendido entre 1 y 9 y con otra fecha cuyo mes esté comprendido entre enero (mes 1) y septiembre (mes 9).

Conviene probarlo también con fechas que no sean del calendario, por ejemplo, con la fecha inexistente 20009975. En tal caso no nos debe preocupar cuál es el resultado escrito por pantalla sino que, simplemente, el programa termine normalmente.

El proyecto **CambioMoneda**, conviene ser probado con los ejemplos del enunciado y con las siguientes cantidades de euros: *0.00*, *0.99* y *1.00*. Cada uno debe calcular antes, a mano o con una calculadora, los resultados que el programa debería dar con las cantidades de *0.99* y *1.00* euros y no dar directamente por buenos los que el programa proporcione.

El proyecto **Cajero** conviene probarlo además de con el ejemplo del enunciado, con otro adicional que cada cual determine y con los valores mínimo y máximo que se pueden sacar de un cajero automático: *10* y *600* euros, respectivamente. Si se prueba con valores enteros negativos o que no sean múltiplos de 10, el resultado mostrado por pantalla no importa, lo único que interesa es que el programa termine normalmente.

El proyecto **Tiempo**, se debiera probar, como mínimo, con los ejemplos del enunciado y con el valor de duración *0*.

El proyecto **Romano**, se puede probar con todos los valores comprendidos entre *1* y *10*. Si se prueba con valores tales como el 0, números negativos o superiores a 10, el resultado mostrado por pantalla no importa, lo único que interesa es que el programa termine normalmente.

El proyecto **Trigonometria**, se debiera probar, como mínimo, además de con los ejemplos del enunciado, con `0º 0’ 0"`, `90º 0’ 0"` y con `359º 59’ 59"`.

### 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 **practica_01** con los siguientes proyectos ubicados en la carpeta `Programacion1/Practica1`:
    - Proyectos C++ **Bienvenida**, **Circunferencias**, **Circulo** y **Formatear**
2. Área de trabajo «Práctica 2» con los siguientes proyectos ubicados en la carpeta `Programacion1/Practica2`:
    - Proyecto C++ **Cuadrados**
    - Proyecto C++ **Fecha**
    - Proyecto C++ **CambioMoneda**
    - Proyecto C++ **Cajero**
    - Proyecto C++ **Tiempo**
    - Proyecto C++ **Romano**
    - Proyecto C++ **Trigonometria**