Prácticas de Programación 1

Grado en Ingeniería Informática

Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

Miguel Ángel Latre y Javier Martínez
Área de Lenguajes y Sistemas Informáticos
Departamento de Informática e Ingeniería de Sistemas

Universidad
Zaragoza

Curso 2015-16

Presentación

Las prácticas de la asignatura **Programación 1** no deben imaginarse como una sucesión de sesiones en laboratorio a las que el alumno debe acudir *a que le cuenten cosas*. Nada más lejos de la realidad. Las prácticas de **Programación 1** comprenden un conjunto de trabajos prácticos de programación que cada alumno debe realizar con bastante autonomía a lo largo del cuatrimestre para alcanzar los resultados de aprendizaje que se describen en la guía docente de la asignatura (http://titulaciones.unizar.es/asignaturas/30204/index15.html).

La información de este manual está organizada en seis capítulos:

- En cada capítulo se proponen diversos trabajos de programación para que cada alumno los realice en un periodo aproximado de dos semanas. Dentro de este periodo tendrá la oportunidad de acudir a una sesión de prácticas en laboratorio en la que un profesor hará el seguimiento del trabajo por él realizado, le brindará su ayuda para resolver los problemas y dificultades con los que se haya topado y, eventualmente, evaluará el trabajo realizado.
- En cada capítulo se dan indicaciones sobre cómo llevar a cabo los trabajos propuestos y, en su caso, se describen, las nuevas herramientas ligadas a la tecnología C++ que conviene utilizar.
- Cada alumno debe tener claro que el trabajo de prácticas es un trabajo que le corresponde realizar a él mismo, como una parte fundamental del aprendizaje de la asignatura. El profesor le puede ayudar, pero esta ayuda es inútil si él no ha trabajo previamente lo suficiente.

Este manual es una guía inicial a las prácticas de la asignatura. Cada alumno deberá acostumbrase a acudir frecuentemente y consultar las siguientes fuentes de información, a las que se puede acceder a través de **página web de la asignatura**, http://webdiis.unizar.es/asignaturas/PROG1/:

- En la sección de ***Documentación C++*** se presentan los enlaces a las páginas web con la documentación de las bibliotecas predefinidas en C++.
- En la sección de ***Materiales Docentes Comunes*** se facilitará código y datos para el desarrollo de algunos de los trabajos propuestos en estas prácticas. Este material podrá ser descargado y copiado.

La siguiente tabla muestra el calendario previsto de sesiones prácticas de **Programación 1** correspondiente al curso 2015-16.

Calendario de sesiones prácticas de PROGRAMACIÓN 1. Curso 2015-16				
Sesiones	**Lunes A**	**Martes A**	**Lunes B**	**Martes B**
1ª	L-28-SEP	M-29-SEP	L-05-OCT	M-06-OCT
2ª	L-19-OCT	M-20-OCT	L-26-OCT	M-27-OCT
3ª	V-06-NOV	M-03-NOV	L-09-NOV	M-10-NOV
4ª	L-16-NOV	M-17-NOV	L-23-NOV	M-24-NOV
5ª	L-30-NOV	M-01-DIC	X-09-DIC	M-15-DIC
6ª	L-21-DIC	M-12-ENE	L-11-ENE	J-07-ENE

En las prácticas de **Programación 1** se desea evitar que el alumno se dedique a explorar el extenso territorio que ofrece un lenguaje como `C++` y acabe perdido en él. Hemos optado porque el alumno solo utilize una parte reducida de lo que el lenguaje `C++` ofrece y, eso sí, lo haga bien, dominando y comprendiendo en todo momento lo que programa. La parte del lenguaje `C++` que está permitido utilizar coincide esencialmente con los elementos presentados y utilizados en el curso de **Programación 1**.

A programar se aprende programando. De ahí la importancia de que cada alumno empieze a programar desde el primer día del curso. ***Programar*** es comprender el problema de tratamiento de información a resolver. ***Programar*** es analizar ese problema, dedicándole el tiempo que sea necesario, hasta decidir cómo abordar su resolución. ***Programar*** es escribir ***en una hoja de papel*** el algoritmo a aplicar para resolverlo. ***Programar*** es trasladar ese algoritmo a código C++, editarlo, compilarlo y ejecutarlo. ***Programar*** es someter nuestros programas a un completo juego de pruebas hasta que tengamos la convicción de que nuestro código no sólo está libre de burdos errores, sino que su comportamiento satisface todas las especificaciones planteadas en el problema de partida.

Todas estas tareas que acabamos de señalar son parte del trabajo de un programador y su realización en estas prácticas es responsabilidad personal de cada alumno.

Zaragoza, Septiembre de 2015

Miguel Ángel Latre y Javier Martínez
Departamento de Informática e Ingeniería de Sistemas
de la Universidad de Zaragoza

Práctica 1: Cómo desarrollar programas escritos en C++

1.1. Objetivos de la práctica

Esta primera práctica de la asignatura persigue los siguientes objetivos:

- Aprender a utilizar los puestos de trabajo disponibles en los laboratorios de programación.
- Aprender a utilizar un IDE (*Integrated Development Environment* o Entorno de Desarrollo Integrado) para el desarrollo y puesta a punto de programas `C++`.
- Estudiar el comportamiento de algunos programas `C++` elementales, las instrucciones utilizadas en ellos y programar en ellos algunas pequeñas modificaciones.

Es esencial que todos los alumnos asistan y completen el trabajo propuesto en esta práctica, ya que lo que se aprende en ella deberá ser utilizado a partir de ese momento de forma continuada.

Los alumnos que no hayan formalizado aún su matrícula en la asignatura también deben realizar la práctica. Si a un alumno le resultara imposible realizar la práctica con su grupo de prácticas, deberá realizarla con cualquiera de los restantes grupos, sin importar si se trata de un grupo de mañana o de tarde.

El desarrollo de la sesión correspondiente a esta práctica tiene dos partes diferenciadas.

1. En la primera parte cada alumno deberá atender las explicaciones del profesor sobre los tres primeros apartados de este quión y seguir sus instrucciones.
2. En la segunda parte cada alumno deberá desarrollar individualmente el trabajo propuesto en el apartado *1.4 Trabajo a desarrollar en esta práctica*. Para ello conviene que, antes de acudir al laboratorio a participar en la sesión de prácticas, haya leído atentamente el guión completo de la práctica, haya comenzado a preparar el trabajo descrito en el apartado 1.4 y haya estudiado el contenido del apartado *1.5 Manipuladores para dar formato a los datos de salida*.

El trabajo asociado a esta práctica concluye nada más acabar la primera sesión de prácticas en laboratorio ya que al día siguiente hay que empezar a trabajar la práctica siguiente.

1.2. Los puestos de trabajo

Cada uno de los puestos de trabajo de los laboratorios de programación consta de un computador personal conectado a la red de comunicaciones de la Universidad de Zaragoza.

Si el computador está apagado y lo ponemos en funcionamiento, nos permite seleccionar el sistema operativo con el que deseamos trabajar. En diferentes asignaturas utilizaremos normalmente uno de los sistemas operativos siguientes:

- **CENT OS**. Sistema operativo derivado del Linux RHEL (*Red Hat Enterprise Linux*). Para trabajar con él hay que disponer de una cuenta de usuario. Cada alumno dispone de una cuenta de usuario abierta por el administrador del sistema, una vez ha formalizado su matrícula en los estudios. Una vez cargado el *CENT OS*, el usuario debe identificarse mediante su nombre de usuario (*username*) y su contraseña (*password*). Por ejemplo:

  ```
  username: A333444
  password: miClave856
  ```

  Tras la escritura del nombre de usuario y, antes de escribir la contaseña, se tiene la posibilidad de seleccionar el idioma.

- **Windows XP**. Para trabajar con él se recomienda hacerlo a través de su propia cuenta de usuario. Una vez cargado el sistema, el usuario debe cerrar la sesión de invitado que se abre automáticamente,y proceder a identificarse mediante su nombre de usuario (*username*) y su contraseña (*password*) y debe seleccionar el dominio DIISLAB. Por ejemplo:

  ```
  usuario: A333444
  contraseña: miClave856
  conectarse a: DIISLAB
  ```

El **nombre de usuario** y la **contraseña** para acceder a los dos sistemas operativos anteriores son comunicados a cada alumno de nuevo ingreso en los estudios mediante un mensaje electrónico a su cuenta de correo en la Universidad. El nombre de usuario y, sobre todo, la contraseña de acceso a nuestra cuenta es **personal y debe mantenerse en secreto** para evitar un mal uso de nuestra cuenta por terceros, del cual seríamos corresponsables.

1.2.1. Una sesión de trabajo con el sistema operativo CENT OS

En las prácticas de *Programación 1* trabajaremos con el sistema operativo ***CENT OS***.

Tras acceder a nuestra cuenta y, tras un tiempo de espera prudencial, aparecen en pantalla dos iconos, ***Equipo*** y ***Carpeta personal de Axxxxxx*** (donde ***xxxxxx*** es el número de identificación del alumno o NIA), por ejemplo ***Carpeta personal de A333444***, y tres menús deplegables en su parte superior (***Aplicaciones***, ***Lugares*** y ***Sistema***).

Nos interesa especialmente la carpeta que tiene el nombre de nuestra cuenta, en el ejemplo que estamos presentando ***Carpeta personal de A333444***, ya que en ella podremos crear nuevas carpetas, borrarlas cuando queramos y almacenar en ellas ficheros.

Conviene saber que estas carpetas no se encuentran almacenadas en un disco local del computador personal de nuestro terminal sino en un clúster de computadores *unix* denominado

hendrix, integrado por los computadores ***hendrix01*** y ***hendrix02***, conectados a la red informática de la Universidad de Zaragoza. De este modo, nuestras carpetas y ficheros podrán ser accedidos desde cualquier otro puesto de trabajo de los que dispone el *Departamento de Informática e Ingeniería de Sistemas* (*DIIS*) y se conservarán allí hasta que decidamos borrarlos.

El clúster ***hendrix*** realiza funciones de servidor de los ficheros de muchos centenares de estudiantes. Dado que su capacidad de almacenamiento es finita, cada usuario tiene limitada la cantidad de información a almacenar. Ello significa que cada usuario debe hacer una gestión adecuada de lo que se almacena, debiendo eliminar sistemáticamente aquellos ficheros que no sean necesarios para trabajar en el futuro.

Desde el menú desplagable ***Applications*** son accesibles diversos programas de aplicación que podemos ejecutar. En las prácticas de esta asignatura solo vamos a tener que ejecutar dos aplicaciones:

- El navegador **Mozilla Firefox** para consultar diversas páginas web, especialmente las siguientes:
  - La web **http://webdiis.unizar.es/asignaturas/PROG1/** de la asignatura Programación 1.
  - La web **http://www.cplusplus.com/** con amplia y diversa documentación sobre el lenguaje `C++`. Dentro de esta página se puede acceder al manual de referencia de la biblioteca estándar `C++` en la dirección **http://www.cplusplus.com/reference/**.
- El entorno de desarrollo integrado ***CodeLite*** (más información de este entorno en ***http://codelite.org/***).

Cuando hayamos concluido nuestra sesión de trabajo deberemos cerrar las ventanas abiertas y salir de nuestra cuenta de trabajo en el sistema operativo ***CENT OS***. Hay que evitar dejarla abierta para impedir que alguien pueda hacer un uso indebido de ella. Para ello debemos seleccionar la orden ***Sistema/Cerrar la sesión de A333444*** situada en el menú desplegable de la parte superior derecha de la pantalla.

1.3. *CodeLite*: un entorno de desarrollo integrado

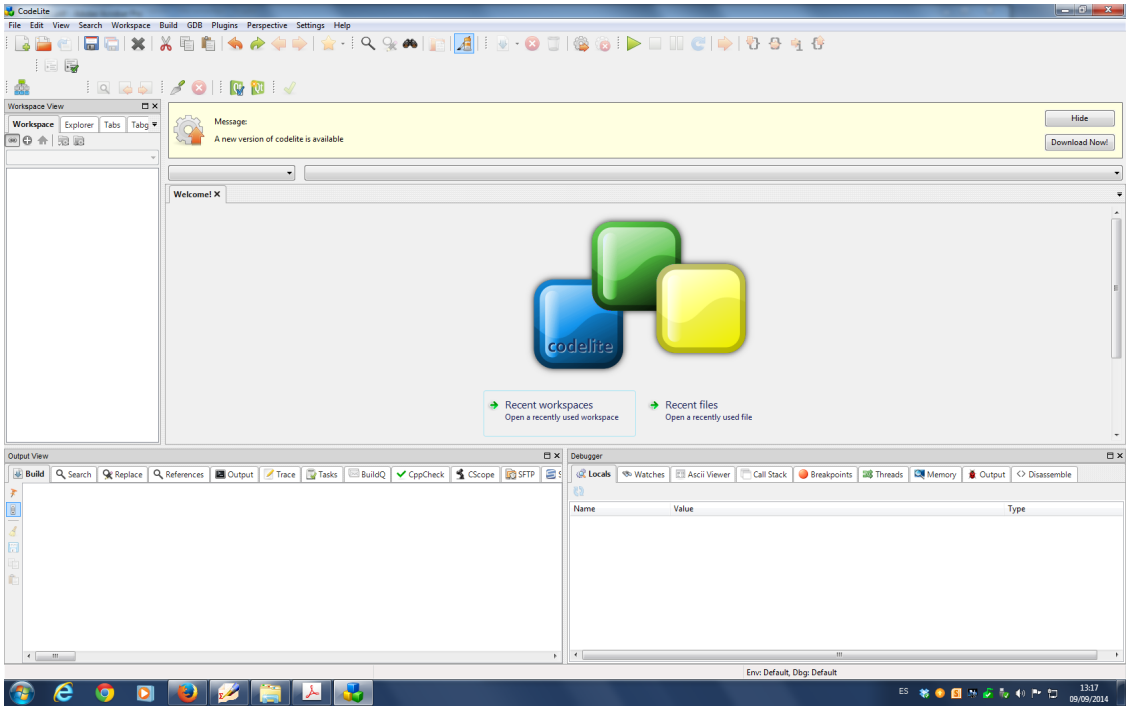
Un entorno de desarrollo integrado (en inglés *integrated development environment* o *IDE*) es un programa informático que integra un conjunto de herramientas que facilitan el diseño, la puesta a punto y el mantenimiento de programas. Dichas herramientas posibilitan la realización de las siguientes tareas:

- Gestionar nuestros proyectos de programación.
- Editar los ficheros fuente con el código de nuestros programas.
- Compilar el código de nuestros programas y facilitar su corrección.
- Informar de errores en el código de nuestros programas.
- Depurar nuestros programas hasta que su comportamiento sea el deseado.
- Ejecutar nuestros programas cuantas veces sea preciso.

A partir de este momento vamos a utilizar este entorno para desarrollar programas `C++`.

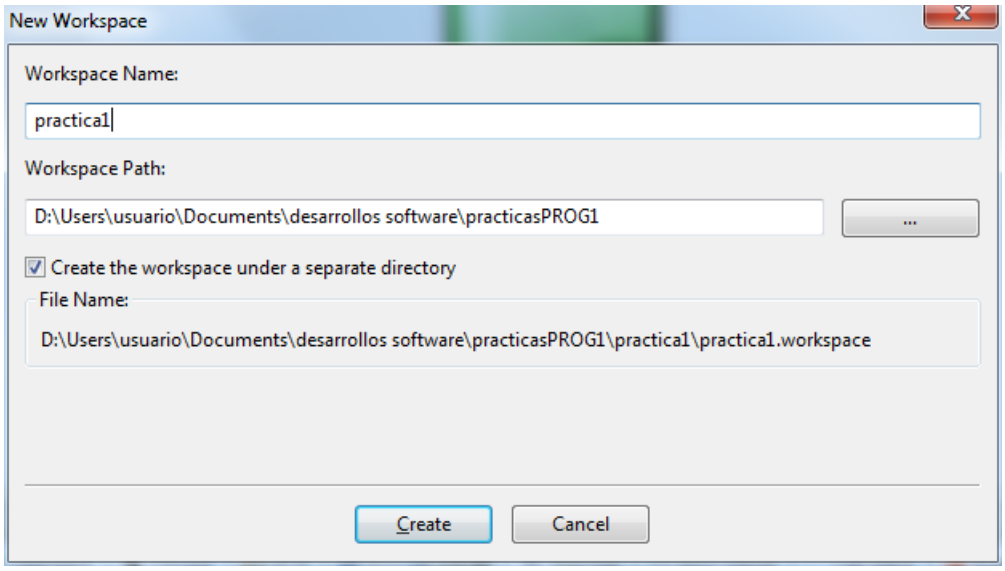
1.3.1. Área de trabajo o *workspace*

Al ejecutar la aplicación *CodeLite*, el entorno de programación presenta una ventana de bienvenida posibilitando la apertura de un **área de trabajo** o ***workspace***.



Si no disponemos aún de ningún área de trabajo o *workspace* podemos crear una nueva y ubicarla dentro de una carpeta o directorio. Para ello procedemos, en primer lugar, a crear la carpeta ***practicasPROG1*** en la que ubicaremos las sucesivas áreas de trabajo que iremos creando en las prácticas de la asignatura.

Desde el entorno *CodeLite* se pueden crear nuevas áreas de trabajo ejecutando la orden **Workspace/New Workspace**, que puede ser seleccionada desde el menú desplegable situado en la parte superior de la ventana. Al ejecutarla se abre una nueva ventana que permite seleccionar la carpeta en la que ubicar la nueva área de trabajo (en la carpeta ***practicasPROG1***) y el nombre de la nueva área de trabajo, en este caso, ***practica1***.

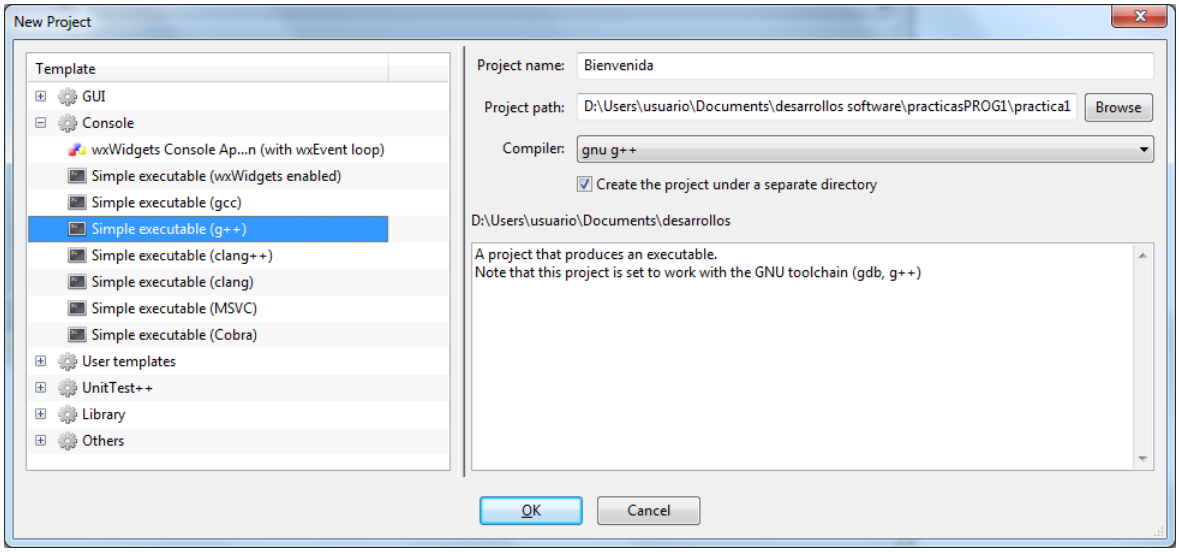


De esta forma se ha creado el área de trabajo ***practica1***, ubicada en la carpeta ***practicasPROG1***.

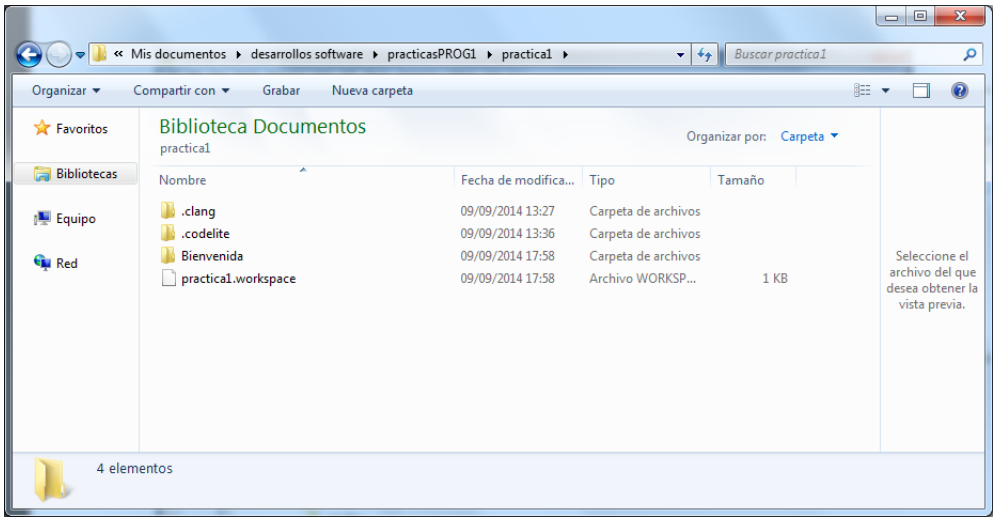
En un área de trabajo se pueden desarrollar varios programas. Cada uno de ellos requiere la definición de un **proyecto**, tal como se explica en el siguiente apartado.

1.3.2. Proyectos de un área de trabajo

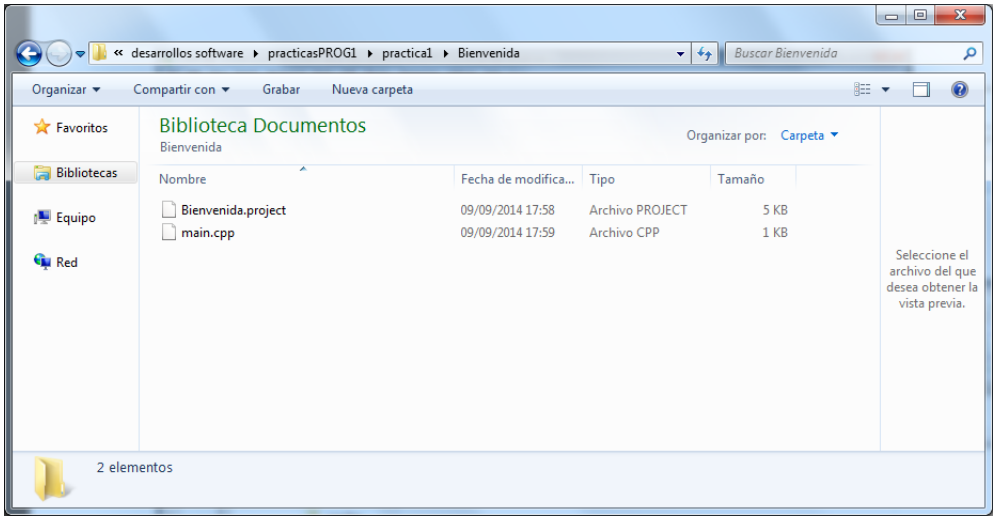
Para definir un nuevo proyecto basta ejecutar la orden **Workspace/New Project**, especificando el nombre del nuevo proyecto y el tipo de proyecto (en este caso, un programa `C++` que interacciona con el operador a través de consola).



Al crear el proyecto *Bienvenida*, en el área de trabajo *practica1*, se crea una nueva carpeta con ese nombre, dentro de la carpeta *practica1*. En la carpeta *practicasPROG1\practica1*, tras haber creado el proyecto *Bienvenida*, se encuentrarían las carpetas correspondientes a los proyectos que hubieremos creado hasta el momento en el área de trabajo, así como un fichero descriptivo del área, el fichero *practica1.workspace*.



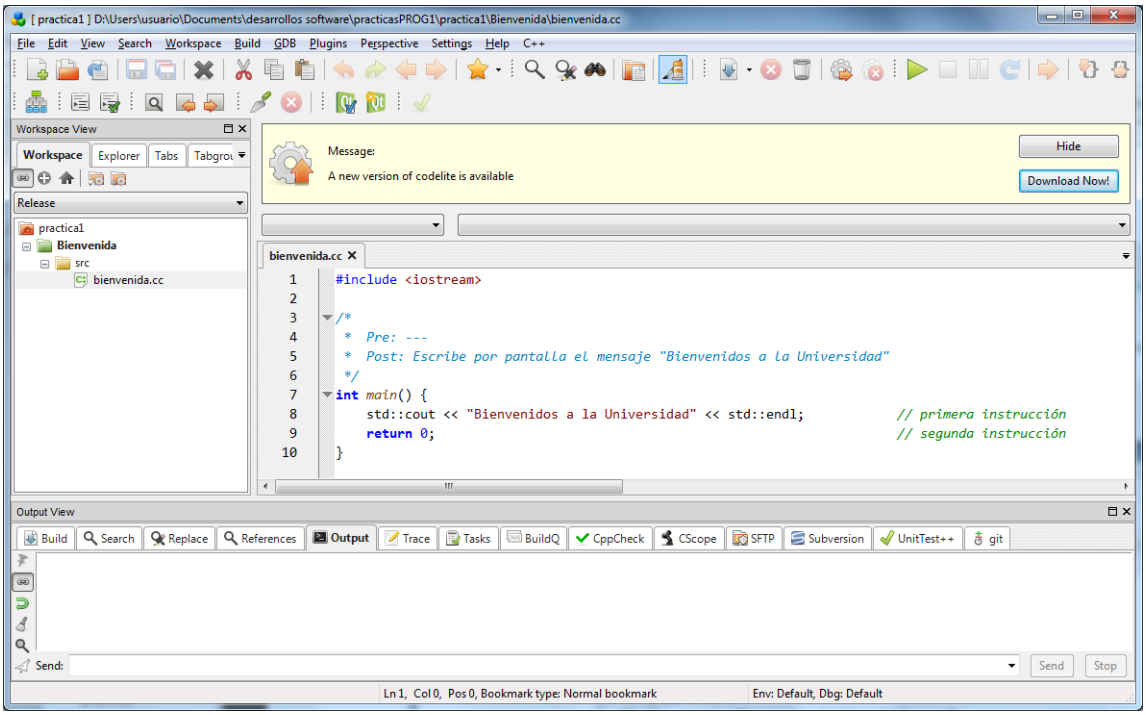
Por su parte, en la carpeta *practicasPROG1\practica1\Bienvenida* se almacenan los ficheros que constituyen su código, así como un fichero descriptivo del proyecto, el fichero *Bienvenida.project*. El contenido de esta carpeta se muestra a continuación.



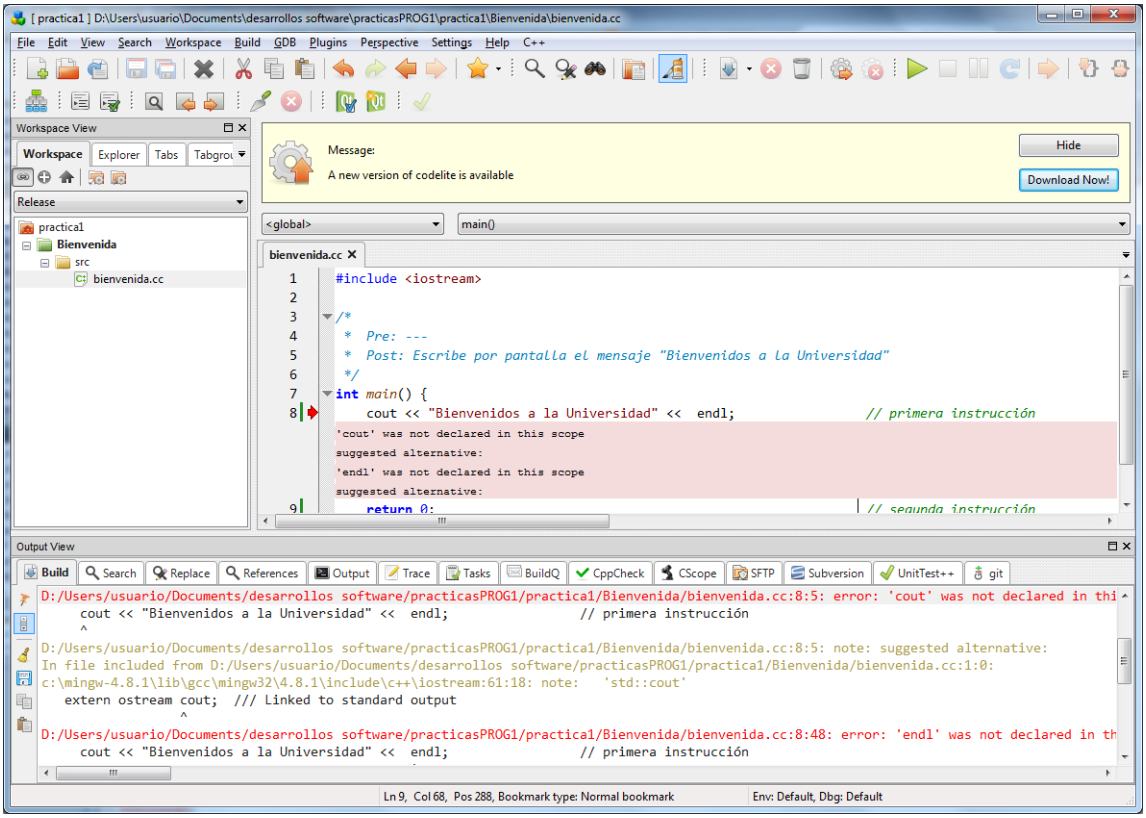
1.3.3. Puesta a punto de un programa C++ en el IDE *CodeLite*

Para desarrollar un programa C++ en *CodeLite* hay que seguir los siguientes pasos :

- Editar el código fuente de los ficheros que integran el programa. En el ejemplo que estamos considerando, cambiaremos el nombre del fichero **main.cpp** por el nombre **bienvenida.cc** y lo editaremos copiando el código del primer programa presentado en el capítulo 1 del texto de Programación 1.

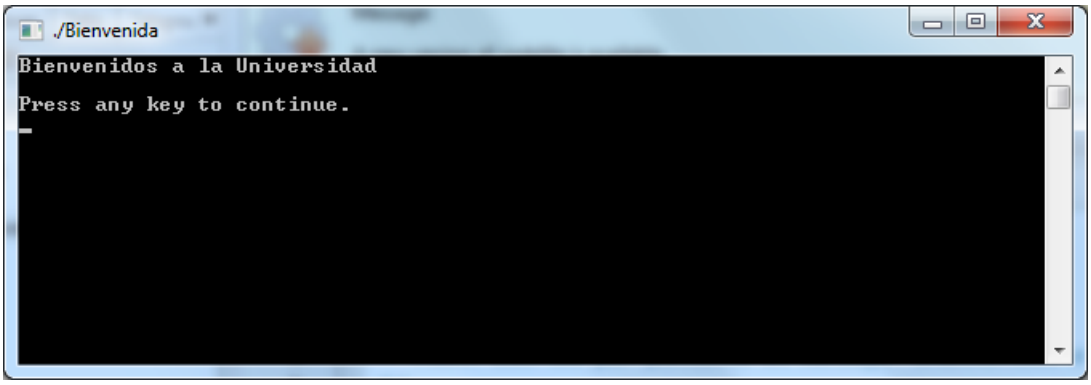


- Compilar el código fuente de los ficheros que integran el programa ejecutando la orden **Build/Build Project**. En el caso de que el compilador señale la existencia de errores, éstos deberán ser corregidos antes de proseguir. Los errores son advertidos mediante mensajes en pantalla que informan del error.



- Ejecutar el programa compilado seleccionando la orden **Build/Run** o pulsando la pequeña fecha de color naranja orientada hacia la derecha, situada en la parte superior derecha de

la ventana. Se creará una ventana, similar a la que se presenta a continuación, que muestra el resultado de ejecutar el programa.



1.4. Trabajo a desarrollar en esta práctica

Se propone que cada alumno realice las tareas que se describen en los apartados siguientes.

Alguno de los fragmentos de código que se mencionan a continuación pueden ser descargados desde la sección de ***Materiales Docentes comunes*** de la web de esta asignatura: **http://webdiis.unizar.es/asignaturas/PROG1/**.

1.4.1. Definir la estructura de un área de trabajo para el desarrollo de programas C++

Crear una estructura organizativa para trabajar con el *IDE CodeLite*. Para ello se deberán ejecutar las siguientes tareas.

- **Tarea 1**. Crear un área de trabajo o *workspace*. Para ello cada alumno debe dar los siguientes pasos:
  - Debe crear una carpeta de nombre **practicasPROG1** en la que ubicar los proyectos a desarrollar en cada una de las prácticas de la asignatura.
  - Debe crear una primera área de trabajo (*workspace*) denominada **practica1** que estará ubicada en la carpeta **practicasPROG1**. En este área se desarrollarán los programas correspondientes a la primera práctica. Para cada una de las prácticas posteriores se creará en semanas posteriores, en la misma carpeta, una nueva área de trabajo: **practica2**, **practica3**, etc.
  - Debe crear un primer proyecto el área de trabajo **practica1**, el proyecto **Bienvenida** con el programa comentado en los apartados anteriores. Debe compilar su código y, si no se han producido errores, ejecutarlo. Mientras se observen fallos en su compilación o en su ejecución, debe tratar de corregirlos hasta que el comportamiento del programa sea el deseado.
- **Tarea 2**. Debe crear como nuevos proyectos dentro área de trabajo **practica1** los proyectos **Circunferencia** y **Circulo**, para el desarrollo de los restantes programas descritos en el capítulo 1 del texto de la asignatura. Deberá compilar y, cuando no haya errores de compilación, ejecutar ambos programas hasta tener la convicción de que sus comportamientos son los esperados.

- **Tarea 3**. Debe crear un nuevo proyecto, al que debe denominar **Formatear**, dentro área de trabajo **practica1**. El código inicial del programa a desarrollar en él se muestra a continuación.

```
#include <iostream>
#include <iomanip>
#include <cmath>

using namespace std;

/*
 * Pre: desde<=hasta
 * Post: Presenta por pantalla una tabla con una línea por cada uno de los valores
 *       del intervalo [desde,hasta]. En cada línea se muestra un valor, su cuadrado
 *       y su cubo
 */
void mostrarPotencias (int desde, int hasta) {
    cout  << "x" << "x^2" << "x^3" << endl;
    for (int x = desde; x<=hasta; x = x + 1) {
        cout << x << x*x << x*x*x << endl;
    }
}

/*
 * Pre: desde<=hasta
 * Post: Presenta por pantalla una tabla con una línea por cada uno de los valores
 *       de los ángulos, expresados en grados: desde, desde + 5, desde + 10, etc,. que
 *       sean inferiores o iguales a hasta. En cada línea se muestra el valor del ángulo
 *       en grados y en radianes y los valores de sus funciones seno y coseno.
 */
void mostrarAngulos (int desde, int hasta) {
    const double PI = 3.141592653589793;
    cout  << "grados" << "radianes" << "seno" << "coseno" << endl;
    for (double x = desde; x<=hasta; x = x + 5.0) {
        double radianes = PI * x / 180.0;
        cout << x  << radianes << sin(radianes) << cos(radianes) << endl;
    }
}

/*
 *  Pre: ---
 *  Post: Presenta por pantalla una tabla de potencias (en cada línea un valor, su cuadrado
 *        y su cubo) y una tabla de ángulos (en cada línea un ángulo en grados, en radianes,
 *        su seno y su coseno)
 */
int main() {
    /* Presenta la tabla de potencias de los enteros comprendidos entre 5 y 15 */
    mostrarPotencias(5,15);
    /* Presenta una tabla con los ángulos comprendidos entre 0 y 180 grados */
    mostrarAngulos(0,180);
    return 0;
}
```

Este programa adolece de un grave defecto: la presentación de resultados por pantalla es penosa.

El trabajo a realizar por cada alumno es modificar el diseño de las funciones `mostrarPotencias(desde,hasta)` y `mostrarAngulos(desde,hasta)` para que, al ser

invocadas, presenten de forma digna y legible sus resultados.

Se recomienda intentar diferentes presentaciones jugando con varios elementos: la anchura de las columnas de la tabla, la alineación a izquierda o derecha de los datos de cada línea, el número de decimales de los número reales, etc.

Para ello se ha de hacer uso de algunas de las **funciones de formato** o **manipuladores** que se presentan en el siguiente apartado.

El objetivo de esta tarea es aprender a presentar resultados después de conocer qué manipuladores hay disponibles en `C++` y cuál es efecto de cada uno de ellos. Se recomienda consultar la documentación sobre ellos disponible en el manual de referencia de la bibloteca predefinida en `C++`, accesible desde la página web de la asignatura.

Las tareas anteriores pueden intentarse antes de acudir al laboratorio a la sesión correspondiente a esta práctica. Durante la sesión podrá completarse y, en su caso, mejorarse el trabajo previo realizado.

Cada alumno debe haber completado las tareas definidas en esta primera sesión de prácticas en la propia sesión o, en caso necesario, uno o dos días después, ya que debe comenzar cuanto antes a desarrollar el trabajo asociado a la segunda práctica.

1.5. Manipuladores para dar formato a los datos de salida

Un **manipulador** `C++` es una función que se aplica al flujo de datos dirigido hacia la pantalla del operador, o hacia otro dispositivo de salida, para dar formato a los datos que se presenten por ella.

En la primera lección de la asignatura se ha hecho uso de diversos manipuladores en los programas `C++` que se han presentado. Conviene hacer una lectura atenta de ella antes de abordar el trabajo propuesto en esta práctica.

Un resumen de algunos de los manipuladores de uso más común se presenta a continuación:

- Manipuladores definidos en la biblioteca <***iostream***>, concretamente en la biblioteca <***ios***> declarada en ella:
  - **endl**: vacía el *buffer* asociado a la pantalla presentando su contenido por pantalla y finaliza la línea en curso.
  - **flush**: vacía el *buffer* asociado a la pantalla presentando su contenido por pantalla sin finalizar la línea en curso.
  - **dec**: convierte los datos numéricos a base decimal (base 10).
  - **oct** convierte los datos numéricos a base octal (base 8).
  - **hex**: convierte los datos numéricos a base hexadecimal (base 16).
  - **left**: presenta los datos de forma que los caracteres que no sean de relleno se alinean a la izquierda del campo.
  - **right**: presenta los datos de forma que los caracteres que no sean de relleno se alinean a la derecha del campo.
  - **internal**: presenta los datos numéricos de forma que el signo y los caracteres indicativos de la base están alineados a la izquierda del campo y las cifras significativas a la derecha.

- **showpos**: se muestra el signo (+) en los valores positivos.
- **scientific**: establece un modo de trabajo de forma que los datos reales son presentados en notación científica de coma flotante (ej. `1.7234e+01`).
- **fixed**: presenta los datos reales en notación normal de coma flotante (ej. `17.234`).
- **boolalpha**: establece un modo de trabajo de forma que los datos lógicos o booleanos son extraidos de un flujo o añadidos a un flujo como una secuencia de caracteres (`true` o `false`).
- **noboolalpha**: establece un modo de trabajo de forma que los datos lógicos o booleanos son extraidos de un flujo o añadidos a un flujo como valores enteros: `1` en caso de valores lógicos `true` o `0` en caso de valores lógicos `false`.

- Manipuladores definidos en la biblioteca <***iomanip***>

  - **setw**(**int** w): establece la anchura mínima de campo de los datos a presentar por pantalla, es decir el número mínimo de caracteres a presentar completando el dato, en su caso, con el caracter de relleno definido en ese momento.
  - **setfill**(**char** ch): establece `ch` como carácter de relleno.
  - **setprecision**(**int** p): establece el número de cifras significativas de los datos numéricos o, en su caso, el número de cifras decimales; su valor por defecto es 6.

  Un manipulador sólo afecta al dispositivo de entrada o de salida (**cin**, **cout**, etc.) al que se aplica. El efecto de los manipuladores permanece en el dispositivo de entrada o de salida correspondiente hasta que se aplica otro manipulador que lo modifica, a excepción del manipulador **setw**(**int** n) que hay que invocarlo antes de cada dato al que se le quiere definir un ancho de campo.

Más información y algunos ejemplos ilustrativos sobre los manipuiladores disponibles en las bibliotecas predefinas en `C++` se puede consultar en **www.cplusplus.com/reference**

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

2.1. Objetivos de la práctica

El objetivo de la 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** integrada en el entorno **CodeLite**.
- 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 presentación de información suministrada por el operador.

  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 almacenado en un dispositivo tipo *pendrive* o de un modo alternativo, por ejemplo, accediendo por interner a un mensaje que tenga adjuntos los ficheros con dichos códigos.

  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 segumiento del trabajo realizado por cada alumno.

  Los seis programas planteados en esta práctica deben estar desarrollado y accesibles en la cuenta ***hendrix*** de cada alumno antes de finalizar la semana en la que cada alumno tiene su sesión de prácticas.

2.2. Trabajo a desarrollar

Una observación técnica relativa a la utilización del entorno **CodeLite** antes de describir las tareas que cada alumno debe completar durante las dos semanas dedicadas a esta práctica. Debe evitarse nombrar las áreas de trabajo y los proyectos que sean creados desde ***CodeLite*** con nombres que incluyan caracteres con tilde o caracteres especiales como la **ñ**, la **ç**, etc. ya que ello impedirá el correcto acceso del entorno a las carpetas asociadas a ellos.

2.2.1. Primera tarea: Aprendizaje de una herramienta de depuración (*debugger*)

Durante la primera parte de la sesión de prácticas cada alumno pondrá a punto en el área de trabajo **practica2** un nuevo proyecto denominada **TablaMultiplicar** en el cual se desarrollará el programa presentado en el capítulo 4 del texto de la asignatura que presenta por pantalla las tablas de multiplicar que selecciona el operador.

Al código de este programa se puede acceder desde la sección de *Materiales Docentes Comunes* de la web de la asignatura.

Utilizando este programa aprenderemos a crear aplicaciones ejecutables y a utilizar la herramienta **GDB** de depuración del proyecto GNU (o *GDB degugger*) que el IDE **CodeLite** facilita.

Para cada uno de los proyectos definidos en el área de trabajo activa es posible seleccionar el modo de trabajo **Release** (si se desea generar exclusivamente un programa ejecutable) o el modo de trabajo **Debug** (si se desea que el código ejecutable generado permita la depuración del programa). Para hacer esta selección debemos acceder al menú ***Build/Configuration Manager*** de ***CodeLite***.

Creación de una aplicación ejecutable

Si tenemos la certeza de que el programa desarolado está libre de errores y nuestro único objetivo es compilar el programa para obtener una aplicación o programa ejecutable basta proceder como sigue:

1. Seleccionar la configuración **Release** para el proyecto en desarrollo tras seleccionar el menú ***Build/Configuration Manager***. En nuestro caso lo haremos para el proyecto ***TablasMultiplicar***.
2. Compilar el proyecto ***TablasMultiplicar*** ejecutando la orden ***Build/Build Project*** o pulsando la tecla **F7**. La aplicación o programa ejecutable ***TablasMultiplicar*** se almacenará en la carpeta ***Release*** situada dentro de la carpeta propia del área de trabajo **practica2**. Si la carpeta ***Release*** no existía previamente, será creada de un modo automático.
3. Para ejecutar la aplicación o programa ejecutable ***TablasMultiplicar*** basta hacer ***doble click*** en el icono asociado al fichero de la aplicación.
4. El programa también se puede ejecutar desde el entormo ***CodeLite*** ejecutando la orden ***Build/Run***, pulsando la combinación de teclas **Control+F5** o pulsando lel icono que representa una pequeña flecha de color naranja orientada hacia la derecha.

Depuración de un programa. Uso del depurador GDB (*GDB debugger*)

Si nuestro objetivo es depurar un programa para eliminar posible errores y verificar su adecuado comportamiento deberemos proceder como sigue:

1. Seleccionar la configuración **Debug** para el proyecto en desarrollo tras seleccionar el menú ***Build/Configuration Manager***. En nuestro caso lo haremos para el proyecto ***TablasMultiplicar***.

2. Compilar el proyecto ***TablasMultiplicar*** ejecutando la orden ***Build/Build Project*** o pulsando la tecla **F7**. La aplicación o programa ejecutable, con código para la depuración del programa ***TablasMultiplicar***, se almacenará en la carpeta ***Debug*** situada en la carpeta propia del área de trabajo **practica2**. Si la carpeta ***Debug*** no existía previamente, será creada de un modo automático.

3. Estamos en condiciones de ejecutar el programa utilizando el depurador GDB. Para ello disponemos de las siguiente órdenes que serán explicadas y aprendidas en la sesión de prácticas. Estas órdenes pueden ser ejecutadas desde el menú GDB, pulsando una tecla o una combinación de teclas, o haciendo *click* en alguno de los ocho iconos situados en la parte derecha de la línea de iconos situado inmediatamente debajo de los menús.

   - **Orden Start/Continue Debugger** para arrancar y reanudar el proceso de depuración. En un programa se pueden insertar puntos de parada (**breakpoints**). En tal caso la ejecución del programa se detendrá al alcanzar el siguiente punto de parada (***breakpoint***). Al producirse la detención se puede observar el punto de ejecución, señalado mediante una flecha verde a la izquierda del código, y los valores que toman los datos visibles mostrado en una ventana auxiliar.
   - **Orden Pause Debugger** para hacer una pausa en el proceso de depuración, cuando ello fuera necesario.
   - **Orden Restart Debugger** para reiniciar el proceso de depuración.
   - **Orden Stop Debugger** para finalizar el proceso de depuración.
   - **Orden Next** y **Orden Next Instruction** para ejecutar paso a paso el programa.

Más información sobre el depurador GDB (*GDB debugger*) puede consularse en **http://codelite.org/LiteEditor/Documentation#toc6**

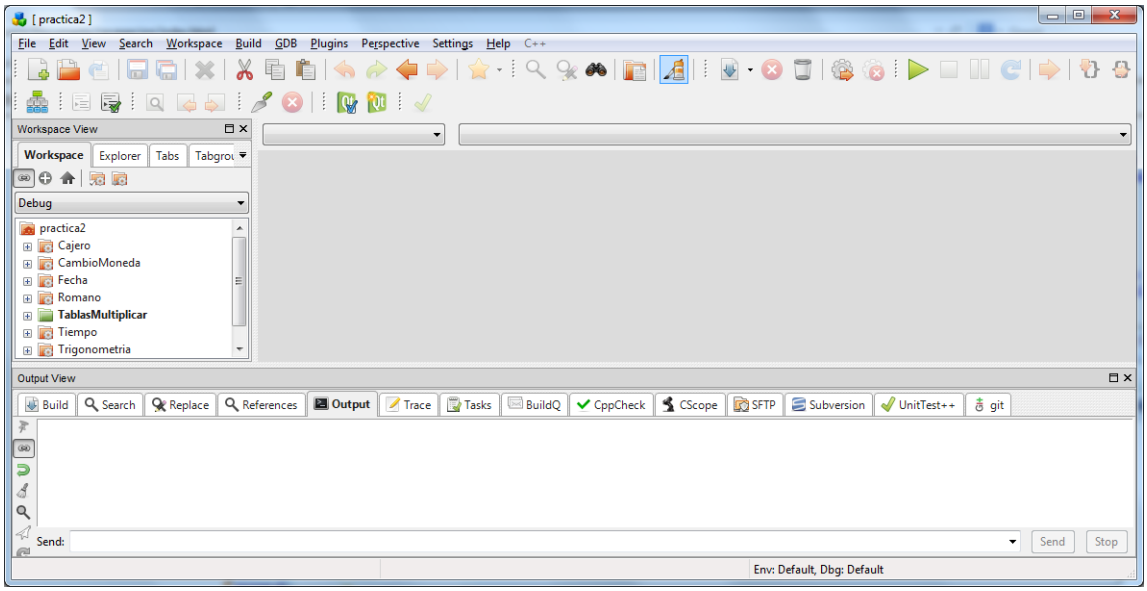
En la primera parte de la sesión de esta práctica se aprenderá a utilizar esta herramienta de depuración. El profesor irá dando instrucciones para guiar el proceso de depuración del programa.

2.2.2. Segunda tarea: desarrollo de programas elementales escritos en C++

Los problemas que aquí se proponen requieren transformar unos determinados datos proporcionados por el operador del programa 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 desarrollados para resolver estos seis problemas se ubicarán en el área de trabajo o *workspace* ***practica2***. En ella se crearán 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 práctica 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.

Proyecto Fecha. Programa de cambio de formato de una fecha

Debe desarrollarse un programa **C++** que interaccione con el operador 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**.

Un segundo ejemplo para ilustrar el comportamiento deseado del programa:

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

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 redondeada.

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

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.

Proyecto Tiempo. Programa sobre la medida del tiempo

El programa interactivo a desarrollar pide al operador que exprese en segundos el valor del tiempo de duración de un determininado evento para, a continuación, informar por pantalla 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 una cantidad de segundos redondeada al segundo más próximo.

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.

A continuación 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
```

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 del ángulo en radianes, así como del valor 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
```

Realización sistemática de pruebas

El trabajo de 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 se hayan realizados pruebas de funcionamiento con los mismos. El objeto de dichas pruebas es el de localizar errores y fallos de funcionamiento, para corregir a continuación sus causas en el código fuente.

En los siguientes párrafos se dan algunas pautas para la realización de pruebas sobre cada uno de los programas de la práctica 2. Recuerda que el objetivo de hacer pruebas es encontrar defectos en los programas realizados, por lo que no deben darse directamente por buenas las respuestas que tus programas den cuando los estés probando, sino que, en los casos no triviales, debes haber calculado antes, a mano o con una calculadora, las respuestas que tus programas deberían proporcionar.

El proyecto **Fecha**, además de probarlo con los datos del ejemplo del enunciado, debes probarlo con una fecha cuyo día esté comprendido entre 1 y 9 y con otra fecha cuyo mes esté comprendido entre enero y septiembre.

El proyecto **CambioMoneda**, debe ser probado con los ejemplos del enunciado y con las siguientes cantidades de euros: *0.00*, *0.99* y *1.00*. Recuerda calcular antes, a mano o con una calculadora, las respuestas que tu programa debería dar con las cantidades de *0.99* y *1.00* euros y no dar directamente por buenas las respuestas que tu programa dé.

El proyecto **Cajero**, pruébalo además de con el ejemplo del enunciado, con otro adicional que tú determines y con los valores mínimo y máximo que se pueden sacar de un cajero automático: *10* y *600* euros, respectivamente.

El proyecto **Tiempo**, pruébalo con los dos ejemplos del enunciado y con el valor de duración *0*.

El proyecto **Trigonometria**, pruébalo, además de con los ejemplos del enunciado, con `0º 0’ 0"`, `90º 0’ 0"` y con `359º 59’ 59"`.

Práctica 3: Diseño y puesta a punto de programas escritos en C++ que trabajan con datos numéricos

3.1. Objetivos de la práctica

Los objetivos de la práctica son los siguientes.

- Aprender a poner a punto programas modulares en C++, desarrollando bibliotecas y los programas C++ que hacen uso de ellas.
- Completar el desarrollo de dos bibliotecas con funciones que facilitan operaciones sobre datos enteros.
- Desarrollar un programa modular dirigido por menú para realizar cálculos y transformaciones de datos enteros.

Al igual que en la práctica anterior, cada alumno debe acudir a la sesión asociada a la práctica con el trabajo ya hecho. La sesión de prácticas debe ser aprovechada para resolver dudas, completar aquellas tareas en la que hayan surgido dificultades no superadas y para que el profesor supervise el trabajo realizado por cada alumno.

3.2. Trabajo a desarrollar

Los programas a desarrollar en esta práctica se van a desarrollar como proyectos ubicados en una nueva área de trabajo, denominada **practica3**, ubicada en la carpeta **practicasPROG1**. Conviene recordar que el nombre del área de trabajo no debe incluir caracteres con tilde.

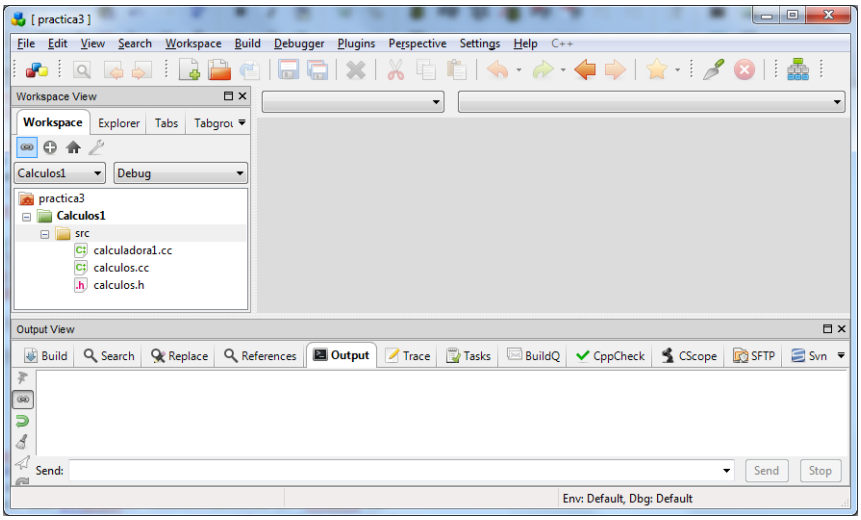
3.2.1. Primera tarea: Puesta a punto de un programa modular

Se debe crear un nuevo proyecto denominado **Calculos1**, que se ubicará en el área de trabajo **practica3**. El proyecto soportará el desarrollo del programa descrito en el capítulo 7 del texto de la asignatura. Se trata de un programa dirigido por menú que permite al operador seleccionar y ejecutar cinco cálculos diferentes con números enteros (calcular el número de cifras de un entero, sumar sus cifras, extraer una cifra, calcular su imagen y comprobar si es primo).

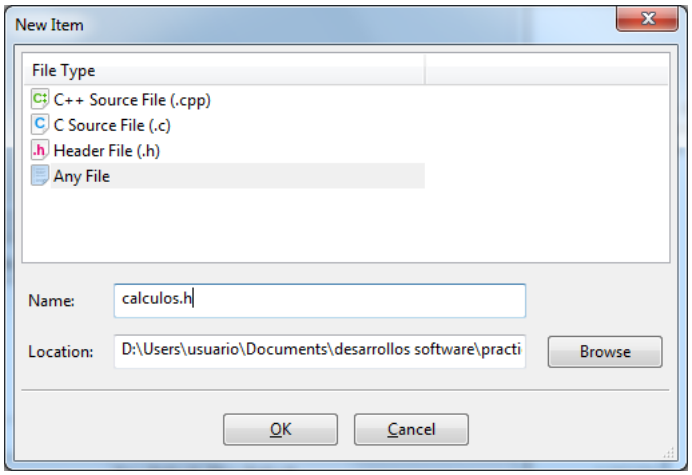
El programa tiene una estructura modular. Consta de un módulo principal y de un módulo de biblioteca. El código de los tres ficheros de que consta el programa se puede copiar a través de la sección de ***Materiales docentes comunes*** de la web de esta asignatura.

- Fichero **calculadora1.cc** con el código del **módulo principal** del programa el cual hace uso de recursos definidos en el módulo **calculos**.
- Fichero **calculos.h** de interfaz del módulo **calculos**.
- Fichero **calculos.cc** de implementación del módulo **calculos**.

En primer lugar se deben crear tres ficheros de código C++ en el proyecto **Calculos1**. Los nombres de estos tres ficheros serán **calculadora1.cc** (el fichero con el módulo *principal* del programa), **calculos.h** (fichero de intezfaz del módulo *calculos*) y **calculos.cc** (fichero de implementación del módulo *calculos*).



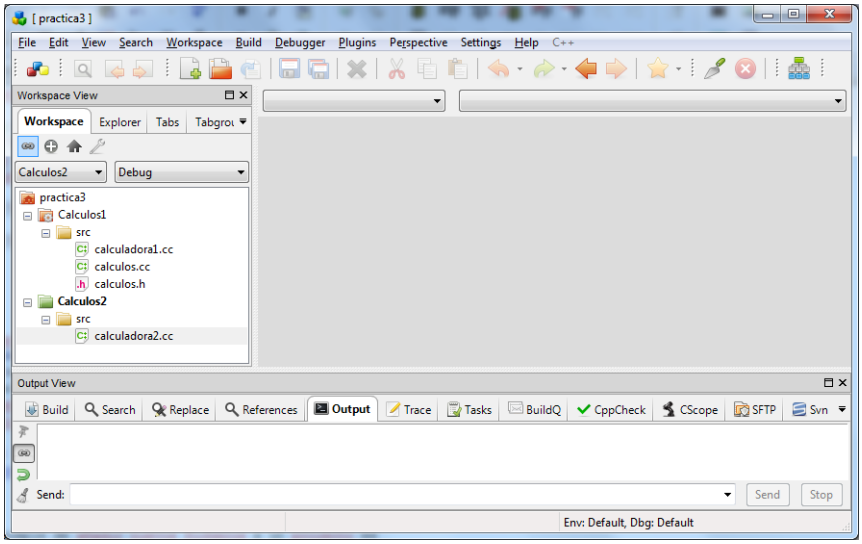
La forma más simple de añadir nuevos ficheros a un proyecto es seleccionar la carpeta **src** mostrada por el entorno **CodeLite** y, pulsando el botón derecho del ratón, ejecutar la orden **Add A New File**. Se abre una nueva ventana en la que deberemos escribir el nombre del nuevo fichero.



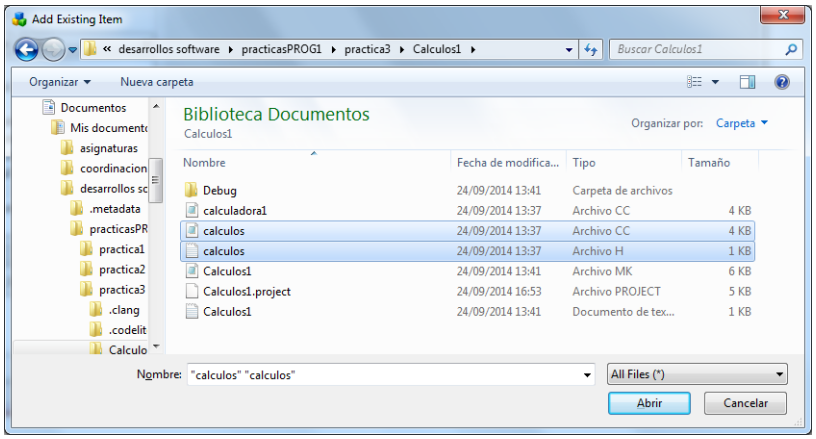
Una vez copiado el código de los tres ficheros procederemos a su compilación y a su ejecución, verificando su correcto comportamiento.

3.2.2. Segunda tarea: Ampliación de la biblioteca y desarrollo de un segundo programa

Se debe crear un nuevo proyecto denominado **Calculos2**, que se ubicará también en el área de trabajo **practica3**. A su módulo principal se le dará el combre **calculadora2.cc**.



Este programa va a hacer uso del módulo **calculos** desarrollado en el proyecto **Calculos1**. No debemos duplicar este módulo, sino declarar que va a ser utilizado en este nuevo proyecto. Para ello seleccionamos la carpeta **src** mostrada por el entorno **CodeLite** y, pulsando el botón derecho del ratón, ejecutamos la orden **Add an Existing File**. Se abre una nueva ventana que nos permite seleccionar los ficheros de interfaz e implementación del módulo **calculos**, ubicados en la carpeta del proyecto **Calculos1**.



Estamos en condiciones tanto de modificar la biblioteca **calculos** desde el proyecto **Calculos2** como de desarrollar un módulo principal para este proyecto. Al escribir el código de éste no debemos olvidar que los dos ficheros del módulo de biblioteca **calculos** están úbicados en la carpeta del proyecto **Calculos1**. Esta circunstancia deberá ser tenida en cuenta al escribir en el código del módulo principal, **calculadora2.cc**, la cláusula **#include** para la inserción del código de dicho módulo de biblioteca.

```
#include "../Calculos1/calculos.h"
```

En esta segunda tarea se pide ampliar las funciones que facilita la biblioteca **calculos** añadiendo en ella las que se especifican a continuación.

```
/*
 *  Pre: ---
 *  Post: Devuelve la cifra más significativa de n cuando n se escribe en base 10
 */
int cifraMasSignificativa (int n);

/*
 *  Pre: ---
 *  Post: Devuelve la cifra de mayor valor de n cuando n se escribe en base 10
 */
int cifraMayor (int n);

/*
 *  Pre: ---
 *  Post: Devuelve un entero cuyas cifras de unidades y decenas, en base 10, coinciden con
 *        las unidades de n en base 10, sus dos siguiente cifras, centenas y millares, en
 *        base 10, coinciden con las decenas de n, en base 10, y así sucesivamente.
 *        Ejemplos:
 *            duplicarCifras(5) = 55
 *            duplicarCifras(-5) = -55
 *            duplicarCifras(7809) = 77880099
 *            duplicarCifras(-7809) = -77880099
 *            duplicarCifras(1002) = 11000022
 *            duplicarCifras(-1002) = -11000022
 */
int duplicarCifras (int n);

/*
 * Pre: n>0
 * Post: Devuelve un entero que al ser escrito en base 10 presenta las cifras no nulas
 *       de [n] y en el mismo orden y, entre cada par de cifras no nulas consecutivas,
 *       el número devuelto presenta un cero. Ejemplos:
 *          cerificar (7) = 7
 *          cerificar (17) = 107
 *          cerificar (113) = 10103
 *          cerificar (170) = 107
 *          cerificar (1203) = 10203
 *          cerificar (1000203) = 10203
 *          cerificar (912000) = 90102
 *          cerificar (91002000) = 90102
 */
int cerificar (int n);
```

A continuación se pide desarrollar también dentro del proyecto **Calculos2** un nuevo módulo de biblioteca denominado **calculosGen** que facilite a los restantes módulos las funciones que se especifican a continuación.

```
/*
 *  Pre: b>=2
 *  Post: Devuelve el número de cifras de n cuando este número se escribe en base b
 */
int numCifras (int n, int b);

/*
```

```
 *  Pre: b>=2
 *  Post: Devuelve la suma de las cifras de n cuando éste se escribe en base b
 */
int sumaCifras (int n, int b);

/*
 *  Pre: i>=1 y b>=2 y b<=10
 *  Post: Devuelve la i-esima cifra menos significatica de n cuando éste se escribe en
 *        base b
 */
int cifra (int n, int i, int b);

/*
 *  Pre: b>=2 y b<=10
 *  Post: Devuelve la cifra más significativa de n cuando n se escribe en base b
 */
int cifraMasSignificativa (int n, int b);

/*
 *  Pre: b>=2 y b<=10
 *  Post: Devuelve la cifra de mayor valor de n cuando n se escribe en base b
 */
int cifraMayor (int n, int b);
```

Finalmente se pide poner a punto un módulo principal, **calculadora2.cc**, que al ser ejecutado presente un comportamiento análogo al del programa desarollado en el proyecto **Calculos1**. Es decir, este nuevo programa debe permitir al operador seleccionar y ejecutar cualquiera de los cálculos facilitados por las tres nuevas funciones añadidas al módulo **calculos** y por las seis funciones definidas en el nuevo módulo **calculosGen**. Cada alumno tiene libertad para decidir los detalles sobre el comportamiento de este programa, procurando siempre de que su utilización tan simple como sea posible.

3.2.3. Metodología de desarrollo y realización de pruebas

El trabajo de desarrollo de un programa no concluye hasta que se han realizado las pruebas necesarias para validar su buen comportamiento, tal como se aprendió en la práctica anterior.

El programa a desarrollar en el proyecto **Calculos2** no estará acabado hasta no haber realizado una prueba exhaustiva del correcto funcionamiento de cada una de las funciones añadidas al módulo **calculos**, de las definidas en el módulo **calculosGen** y del programa en su conjunto.

En un proyecto como éste no conviene escribir el código completo de los tres módulos antes de empezar a realizar pruebas. Por el contrario, conviene realizar un desarrollo incremental. Por ejemplo, desarrollar una función detrás de otra, y no pasar a la siguiente función hasta que el correcto comportamiento de la anterior está suficientemente verificado. Conforme se vayan desarrollando nuevas funciones el programa principal irá evolucionando permitiendo su selección.

Durante la sesión de prácticas, el profesor facilitará el código de un programa cuyo único objetivo es probar el buen comportamiento de las funciones de los módulos de biblioteca **calculos** y **calculosGen**. Con esta herramienta cada alumno podrá validar el trabajo por él desarrollado y, en su caso, detectar y corregir errores.

Esta técnica de prueba basada en el desarrollo de un programa específico para ello deberá ser tenida en cuenta en los próximos trabajos de programación a desarrollar por cada alumno.

Práctica 4: Diseño y puesta a punto de un módulo de biblioteca C++ que trabaja con tablas de datos

4.1. Objetivos de la práctica

En esta práctica, cada alumno debe desarrollar y probar un módulo de biblioteca, escrito en C++, cuya finalidad es facilitar el desarrollo posterior de diversos programas que permitan analizar los datos de grupos de personas que comparten una característica común: habitar una determinada área geográfica, practicar un mismo deporte, ser clientes de una misma empresa, etc.

La información asociada a cada persona relevante para los análisis que se pretenden abordar se limita exclusivamente a su fecha de nacimiento y a su sexo. Otros datos tales como su nombre, apellidos, domicilio, profesión, estado civil, NIF o NIE, etc, no van a ser considerados en nuestros proyectos software, por el momento.

El trabajo propuesto en esta práctica se complementa con el planteado en la práctica siguiente en la que la información correspondientes a cada grupo de personas se almacenará en ficheros para poder recuperar posteriormente esa misma información.

Al igual que en las prácticas anteriores, a la sesión de laboratorio asociada a esta práctica hay que acudir con el trabajo ya hecho y aprovecharla con un doble objetivo:

- Consultar al profesor allí presente las dudas surgidas y las dificultades que hayan impedido resolver satisfactoriamente los problemas planteados y, con su ayuda, tratar de aclarar ideas y avanzar en la resolución de los problemas surgidos.
- Presentar al profesor el trabajo realizado para que este lo pueda revisar, advertir de posibles errores y defectos y dar indicaciones sobre cómo mejorarlo y, en su caso, corregirlo.

4.2. Trabajo a desarrollar

El trabajo a desarrollar en esta práctica se resume a continuación:

- Desarrollo del módulo de biblioteca **grupo** con funciones de aplicación general que trabajan con tablas de datos que almacenan información de un grupo de personas, concretamente su fecha de nacimiento y su sexo.

- Desarrollo de tantos programas de prueba como sean necesarios hasta asegurar el correcto comportamiento de las diferentes funciones que facilita el módulo de biblioteca **grupo**.

En los subapartados que siguen se dan algunas indicaciones complementarias sobre las tareas anteriores.

4.2.1. Primera tarea. Desarrollo del módulo de biblioteca `grupo`

Se debe crear el proyecto **Grupo** dentro del área de trabajo **practica4** ubicada en la carpeta **practicasPROG1**. En este proyecto se desarrollará un módulo de biblioteca denominado **grupo**. Este módulo ofrecerá a los programadores de otros módulos una colección de funciones básicas para trabajar con tablas de datos que almacenan información sobre la fecha de nacimiento y el sexo de cada una de las personas de un grupo.

Habrá que escribir sus ficheros de interfaz y de implementación. El fichero de interfaz de este módulo se presenta a continuación.

```
/*
 * Fichero de interfaz grupo.h del módulo grupo
 */

/*
 * Este módulo facilita una colección de funciones para trabajar con la
 * información asociada a un grupo de personas. De cada persona interesan
 * únicamente dos datos: su fecha de nacimiento y su sexo.
 * La represención de la información de un grupo de personas se gestiona
 * mediante dos tablas, una tabla de datos de tipo [int] que almacena las
 * fechas de nacimiento de cada persona y una tabla de tipo [bool] que
 * codifica el sexo de cada persona: true (cierto) si es mujer y false
 * (falso) si es hombre.
 * Los datos de una persona determinada se almacenan en elementos de las
 * tablas de fechas de nacimientos y de sexo que tienen índices idénticos.
 */

/*
 * Codificación de fechas: Las fechas a la que se hace referencia en este módulo
 * se representan como datos enteros que, si se escriben en base decimal, constan
 * de 8 cifras, aaaammdd, donde las cuatro más significativas, aaaa, representan
 * el año, las dos cifras que les siguen, mm, representan el mes, y las dos cifras
 * menos significativas, dd, representan el día del mes. Ejemplos de fechas:
 * 20150101 y 20151231, que representan el primer y último dia del año 2015.
 *
 * Codificación del sexo de una persona: mediante un dato booleano; si su valor es
 * cierto (true) denota una mujer y si su valor es falso (false) denota un hombre.
 */

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *    Funciones que trabajan con tablas de fechas                                *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

/*
 * Pre: n > 0 y tFechas[0,n-1] almacena n fechas
 * Post: Devuelve la fecha almacenada en tFechas[0,n-1] más antigua
 */
int fechaMasAntigua (int tFechas[], int n);
```

```
/*
 * Pre: n > 0 y tFechas[0,n-1] almacena n fechas
 * Post: Devuelve el año que más se repite en el grupo de fechas almacenadas
 *       en tFechas[0,n-1]
 */
int anyoMasRepetido (int tFechas[], int n);

/*
 * Pre: n > 0 y tFechas[0,n-1] almacena n fechas
 * Post: Devuelve el mes, un número del intervalo [1,12], que más se repite en el
 *       grupo de fechas almacenadas en tFechas[0,n-1]
 */
int mesMasRepetido (int tFechas[], int n);

/*
 * Pre: n > 0 y tFechas[0,n-1] almacena n fechas
 * Post: Devuelve el día del mes, un valor comprendido en el intervalo [1-31],
 *       que más se repite en el grupo de fechas almacenadas en tFechas[0,n-1]
 */
int diaMasRepetido (int tFechas[], int n);

/*
 * Pre: n > 0 y tFechas[0,n-1] almacena n fechas
 * Post: Devuelve la fecha que más se repite en el grupo de fechas almacenadas
 *       en tFechas[0,n-1]
 */
int fechaMasRepetida (int tFechas[], int n);

/*
 * Pre: n >= 0 y tFechas[0,n-1] almacena n fechas
 * Post: Devuelve el número de fechas almacenadas en tFechas[0,n-1] que
 *       coinciden con la fecha dia/mes/anyo
 */
int cuantos (int tFechas[], int n, int dia, int mes, int anyo);

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *    Funciones que trabajan con tablas de sexos                               *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

/*
 * Pre: n >= 0, tSexo[0,n-1] almacena el sexo de un grupo de personas
 * Post: Devuelve el número de datos de tSexo[0,n-1] que corresponden a hombres
 */
int hombres (bool tSexo[], int n);

/*
 * Pre: n >= 0, tSexo[0,n-1] almacena el sexo de un grupo de personas
 * Post: Devuelve el número de datos de tSexo[0,n-1] que corresponden a mujeres
 */
int mujeres (bool tSexo[], int n);

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *     Funciones que gestionan dos tablas que almacenan las fechas de          *
 *     nacimiento y el sexo de un grupo de personas                            *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

/*
 * Pre: n >= 0, tFechas[0,n-1] almacena las fechas de nacimiento de un grupo de
```

```
 *       personas y tSexo[i] almacena el sexo de la persona de ese grupo cuya fecha
 *       de nacimiento es tFechas[i]. Sean NH y NM el número de hombres y mujeres
 *       del grupo, respectivamente, nacidos en el intervalo de fechas [desde,hasta]
 * Post: Si los parámetros hombres y mujeres son ambos cierto (true) entonces
 *       devuelve NH+NM, si sólo hombres es cierto (true) entonces devuelve NH;
 *       si sólo mujeres es cierto (true) entonces devuelve NM; y si ambos son
 *       falso (false) devuelve 0
 */
int contar (int tFechas [], bool tSexo[], int n, int desde, int hasta,
            bool hombres, bool mujeres);

/*
 * Pre: n >= 0, tFechas[0,n-1] almacena las fechas de nacimiento de un grupo de
 *      personas y tSexo[i] almacena el sexo de la persona de ese grupo cuya fecha
 *      de nacimiento es tFechas[i].
 *      Sean NH y NM el número de hombres y mujeres del grupo
 * Post: Si el parámetro sexo es cierto (true) entonces numero toma el valor MH y
 *       almacena en tSeleccion[0,numero-1] las fechas de nacimiento de las MH mujeres
 *       del grupo y si el parámetro sexo es falso (false) entonces numero toma el
 *       valor NH y almacena en tSeleccion[0,numero-1] las fechas de nacimiento
 *       de los NH hombres del grupo
 */
void seleccionar (int tFechas [], bool tSexo[], int n, bool sexo, int tSeleccion [],
                  int& numero);

/*
 * Pre: n >= 0, tFechas[0,n-1] almacena las fechas de nacimiento de un grupo de
 *      personas y tSexo[i] almacena el sexo de la persona de ese grupo cuya fecha
 *      de nacimiento es tFechas[i].
 *      Sean NH y NM el número de hombres y mujeres del grupo
 * Post: Permuta los datos de tFechas[0,n-1] y de tSexo[0,n-1] de forma que los
 *       datos tFechas[0,NM-1] y de tSexo[0,NM-1] correspondan a las NM mujeres
 *       del grupo y los datos tFechas[NM,n-1] y de tSexo[NM,n-1] correspondan
 *       a los NH hombres del grupo, manteniendo la propiedad de que tSexo[i] almacena
 *       el sexo de la persona de ese grupo cuya fecha de nacimiento es tFechas[i]
 */
void distribuir (int tFechas [], bool tSexo[], int n);

/*
 * Pre: n >= 0, tFechas[0,n-1] almacena las fechas de nacimiento de un grupo de
 *      personas y tSexo[i] almacena el sexo de la persona de ese grupo cuya fecha
 *      de nacimiento es tFechas[i].
 *      Sean NH y NM el número de hombres y mujeres del grupo
 * Post: Permuta los datos de tFechas[0,n-1] y de tSexo[0,n-1] de forma que queden
 *       ordenados según la edad, es decir, tFechas[0] y tSexo[0] almacenarán la
 *       información de la persona más veterana del grupo y tFechas[n-1] y tSexo[n-1]
 *       los de la más joven, manteniendo la propiedad de que tSexo[i] almacena
 *       el sexo de la persona de ese grupo cuya fecha de nacimiento es tFechas[i]
 */
void ordenar (int tFechas [], bool tSexo[], int n);
```

El código del fichero de interfaz del módulo, el fichero `grupo.h`, se puede copiar de la sección de materiales docentes comunes de la web de la asignatura. Este fichero no debe ser alterado.

El código del fichero de implementación debe ser escrito por cada alumno.

4.2.2. Segunda tarea. Desarrollo de programas de prueba

Debe procederse a probar el buen comportamiento del código de cada una de las funciones del módulo de biblioteca **grupo**. Conviene probar cada una de estas funciones a medida que se van desarrollando, sin necesidad de esperar a que estén escritas las restantes.

Para ello se deben crear tantos proyectos diferentes (**Prueba01**, **Prueba02**, etc.) cuantos programas de prueba se vayan a desarrollar. Todos ellos se ubicarán en el área de trabajo **practica4**. Cada alumno tiene libertad para plantear los programas de prueba que considere más adecuados.

Sin embargo, se sugiere escribir programas de prueba que comprueben, al menos, los resultados de las funciones desarrolladas al ser invocadas con tablas que cumplan con las siguientes características:

- Que la tabla tenga el tamaño mínimo exigido por la precondición de la función (0 o 1, según el caso), además de probar con tablas con bastantes datos.
- Para las funciones `fechaMasAntigua(tFechas,n)`, `anyoMasRepetido(tFechas,n)`, `mesMasRepetido(tFechas,n)`, `diaMasRepetido(tFechas,n)` y `fechaMasRepetida(tFechas,n)`, se sugiere realizar tres pruebas adicionales con tablas en las que el dato devuelto por la función se encuentre en la primera componente de la tabla, en una componente intermedia y en la última componente, respectivamente.
- Para la función `cuantos(tFechas,n,dia,mes,anyo)`, se sugiere realizar pruebas en las que la fecha a contar se encuentre en la primera y/o última componente.
- Para las funciones `hombres(tSexo,n)` y `mujeres(tSexo,n)`, se sugiere realizar cuatro pruebas adicionales, en las que la primera y última componentes correspondan respectivamente a hombre y mujer.
- Para la función `contar(tFechas,tSexo,n,desde,hasta,hombres,mujeres)`, se sugiere hacer pruebas en las que `desde` sea mayor que `hasta` y en las que los valores de la primera y última componentes deban y no deban contabilizarse por alguna de las siguientes razones: por estar dentro o no del intervalo definido por `desde` y `hasta`, por ser hombre o mujer y haber establecido que se desea contabilizar solo hombres y por ser hombre o mujer y haberse establecido que se desea contabilizar solo mujeres.
- Para la función `seleccionar(tFechas,tSexop,n,sexo,tSeleccion,numero)`, se sugiere realizar pruebas en la que se seleccionen 0, 1 y todas las personas de las tablas `tFechas` y `tSexo`, además de plantear situaciones de prueba equivalentes a las enumeradas en el punto anterior para la función `contar(tFechas,tSexo,n,desde,hasta,hombres,mujeres)`.
- Para la función `distribuir(tFechas,tSexo,n)`, se sugiere hacer cuatro pruebas en los casos de que en la primera y última componentes haya tanto un hombre como una mujer.
- Para la función `ordenar(tFechas,tSexo,n)`, se sugiere hacer cuatro pruebas en las que en la primera y última componentes se encuentren tanto la persona más joven como la más vieja.

Práctica 5: Diseño y puesta a punto de un módulo C++ de funciones que trabajan con ficheros

5.1. Objetivos de la práctica

El objetivo de esta práctica es aprender a trabajar con ficheros en C++. Para ello se va a trabajar con cuatro tipos de ficheros que almacenan todos ellos información de una secuencia de personas que integran un grupo. Los datos relativos a un grupo de personas se van a poder almacenar en un fichero con uno de los cuatro formatos se presentan a continuación:

- Ficheros que almacenan información de un grupo de personas con un formato textual T01 que responde a la siguiente sintaxis:

```
<fichero_T01> ::=   { <residente>  fin_línea }
<persona> ::= <dia> <sep> <mes> <sep> <año> <sep> <mujer>
<dia> ::= literal_entero
<mes> ::= literal_entero
<año> ::= literal_entero
<mujer> ::= literal_booleano
<sep> ::= " " { " " }
```

  Todos los datos están representados de forma textual, mediante su representación literal. Dos datos consecutivos en una misma línea están separados por uno o más espacios en blanco.

  Ejemplo de un fichero que almacena información de un grupo de personas con formato T01:

```
3   12   1985   TRUE
31   7   1983   FALSE
4    4   1992   FALSE
    .   .   .
9   10   1989   TRUE
1    1   1990   FALSE
31  12   1990   TRUE
```

- Ficheros que almacenan información de un grupo de personas con un formato textual `T02` que responde a la siguiente sintaxis:

```
<fichero_T02> ::=   { <persona>  fin_línea }
<persona> ::= <dia> <sep> <mes>  <sep> <año>  <sep> <mujer>
<dia> ::= literal_entero
<mes> ::= literal_entero
<año> ::= literal_entero
<mujer> ::= literal_booleano
<sep> ::= ","
```

  Todos los datos están también representados mediante su representación literal. Difiere del formato `T01` en que dos datos consecutivos en una misma línea están separados por una coma.

  Ejemplo de un fichero que almacena información de un grupo de personas con formato `T02`:

```
3,12,1985,TRUE
31,7,1983,FALSE
4,4,1992,FALSE
    .  .  .
9,10,1989,TRUE
1,1,1990,FALSE
31,12,1990,TRUE
```

- Ficheros que almacenan información de un grupo de personas con un formato binario `B01` que responde a la siguiente sintaxis:

```
<fichero_B01> ::=   { <persona> }
<persona> ::= <dia> <mes> <año> <mujer>
<dia> ::= dato_tipo_int
<mes> ::= dato_tipo_int
<año> ::= dato_tipo_int
<mujer> ::= dato_tipo_bool
```

  Todos los datos están almacenados en binario, como datos de tipo `int` (día, mes y año) o como dato de tipo `bool` (mujer).

  Ejemplo de un fichero que almacena información de un grupo de personas con formato `B01`, en el que cada dato binario se ha escrito entre ángulos, mediante una representación literal, ya que su auténtica representación binaria resultaría difícil descifrar:

```
<3> <12> <1985> <true> <31> <7> <1983> <false> <4> <4> <1992> <false>
    .  .  .
<9> <10> <1989> <true> <1> <1> <1990> <false> <31> <12> <1990> <true>
```

- Ficheros que almacenan información de un grupo de personas con un formato binario `B02` que responde a la siguiente sintaxis:

```
<fichero_B02> ::=   { <persona> }
<persona> ::=  dato_tipo_Persona
```

Toda la información está almacenada en binario en el fichero, como una secuencia de datos de tipo `Persona`, definido en el módulo de biblioteca `persona`.

Ejemplo de un fichero que almacena información de un grupo de personas con formato `B02`, en el que cada dato binario se ha escrito entre ángulos, mediante una representación literal, separando los campos de cada registro con comas, ya que su auténtica representación binaria resultaria difícil descifrar:

```
<19851203,true> <19830731,false> <19920404,false>
    .   .   .
<198910009,true> <19900101,false> <19901212,true>
```

Al igual que en las prácticas anteriores, a la sesión de laboratorio asociada a esta práctica hay que acudir con el trabajo que se describe a continuación ya hecho y aprovecharla con un doble objetivo:

- Consultar al profesor allí presente las dudas surgidas y las dificultades que hayan impedido resolver satisfactoriamente los problemas planteados y, con su ayuda, tratar de aclarar ideas y avanzar en la resolución de los problemas surgidos.
- Presentar al profesor el trabajo realizado para que este lo pueda revisar, advertir de posibles errores y defectos y dar indicaciones sobre cómo mejorarlo y, en su caso, corregirlo.

5.2. Trabajo a desarrollar

En esta práctica se han de desarrollar varios proyectos que se describen a continuación. Todos ellos se ubicarán en el área de trabajo **practica5**.

5.2.1. Primera tarea. Creación del módulo de biblioteca `persona`

Esta primera tarea consiste en desarrollar un módulo de biblioteca denominado **persona** para definir el tipo de dato **Persona** y facilitar una mínima colección de funciones para trabajar con datos de este tipo.

El módulo de biblioteca **persona** se desarrollará en un proyecto denominado **Persona** dentro del área de trabajo **practica5**.

El código del fichero de interfaz del módulo, `persona.h`, se presentan a continuación. En él se ha introducido la novedad de establecer zonas de compilación condicionada cuyo comienzo y final vienen delimitados por las directivas:

#ifndef `NOMBRE_MACRO`

Zona de código cuya compilación está condicionada

#endif

La zona de código delimitada por ambas directivas solo será compilada en el caso de que `NOMBRE_MACRO` no haya sido definida con anterioridad mediante una cláusula **#define** `NOMBRE_MACRO`.

Imaginemos que el fichero de interfaz del módulo de biblioteca **persona** está incluido, mediante una directiva `#include "persona.h"` en varios de los ficheros de que consta el

programa. La primera vez que el compilador alcance esa zona de código, este será compilado ya que la macro `PERSONA_H` no ha sido definida aún. Ello provocará que se tenga en cuenta la directiva `#define PERSONA_H` que establece la definición de la macro `PERSONA_H`. A partir de ese momento esta zona del código no volverá a ser compilada y, por lo tanto, se evitarán errores de compilación por duplicación de la definición del tipo de dato `Persona` y de las funciones allí definidas.

En el caso del fichero de interfaz del módulo **persona**, esta no es una precaución exagerada, sino una necesidad de diseño, como es fácil de comprobar si se omiten las directivas que facilitan la compilación condicionada antes descrita.

El código del fichero de interfaz del módulo **persona** es el siguiente.

```
/*
 *  Fichero de interfaz persona.h del módulo persona
 */

#ifndef PERSONA_H              // Principio de la zona de compilación condicionada

#define PERSONA_H

/*
 * Definición del tipo de dato Persona
 */
struct Persona {
    int fecha;          /* fecha escrita en base 10 tiene la forma aaaammdd */
    bool mujer;         /* si mujer = true entonces es una mujer sino es un hombre */
};

/*
 * Pre: d>=1 y d<=31, m>=1 y m<=12, a>0
 * Post: Devuelve un dato de tipo Persona que corresponde a una persona
 *       nacida el d/m/a y cuyo sexo viene definido por el valor de
 *       [esMujer] (true -> es una mujer, false -> es un hombre)
 */
Persona definirPersona (int d, int m, int a, bool esMujer);

/*
 * Pre: ---
 * Post: Devuelve la fecha de nacimiento de [p] como un entero que, al ser escrito
 *       en base 10, tiene la forma aaaammdd, donde aaaa: año, mm: mes y dd: dia
 */
int nacido (Persona p);

/*
 * Pre: ---
 * Post: Devuelve true si la persona [p] es mujer y false si es hombre
 */
bool esMujer (Persona p);

#endif              // Fin de la zona de compilación condicionada
```

El código del fichero de interfaz del módulo, el fichero `persona.h`, se puede copiar de la sección de materiales docentes comunes de la web de la asignatura. Este fichero no debe ser alterado.

El fichero de implementación, `persona.cc`, debe ser programado por cada alumno.

5.2.2. Segunda tarea. Creación del módulo de biblioteca `gestionFicheros`

El módulo de biblioteca **gestionFicheros** se desarrollará en un proyecto denominado **GestionFicheros** dentro del área de trabajo **practica5**.

El código del fichero de interfaz, `gestionFicheros.h`, del módulo **gestionFicheros** es el siguiente. Este módulo facilita cuatro funciones para leer información de un grupo de personas almacenada en un fichero atendiendo a los cuatro formatos descritos anteriormente, así como cuatro funciones para crear un fichero que almacene la información de un grupo de personas, aplicando esos mismos cuatro formatos.

```
/*
 * Fichero gestionFicheros.h de interfaz del módulo gestionFicheros
 */

/*
 * En lo que sigue se presenta la sintaxis de cuatro formatos para el
 * almacenamiento en un fichero de los datos de una secuencia de personas
 * que constituyen un grupo. Los datos <dia>, <mes> y <año> definen la fecha
 * nacimiento de la persona y <mujer> define su sexo (mujer si su valor es
 * true [cierto] u hombre si su valor es false [falso])
 *
 * Formato textual T01 para almacenar información de un grupo de personas
 * en un fichero:
 *   <fichero_T01> ::= { <persona> fin_línea }
 *   <persona> ::= <dia> <sep> <mes> <sep> <año> <sep> <mujer>
 *   <dia> ::= literal_entero
 *   <mes> ::= literal_entero
 *   <año> ::= literal_entero
 *   <mujer> ::= literal_booleano
 *   <sep> ::= " " { " " }
 *
 * Formato textual T02 para almacenar información de un grupo de personas
 * en un fichero:
 *   <fichero_T02> ::= { <persona> fin_línea }
 *   <persona> ::= <dia> <sep> <mes> <sep> <año> <sep> <mujer>
 *   <dia> ::= literal_entero
 *   <mes> ::= literal_entero
 *   <año> ::= literal_entero
 *   <mujer> ::= literal_booleano
 *   <sep> ::= ","
 *
 * Formato binario B01 para almacenar información de un grupo de personas
 * en un fichero:
 *   <fichero_B01> ::= { <persona> }
 *   <persona> ::= <dia> <mes> <año> <mujer>
 *   <dia> ::= dato_tipo_int
 *   <mes> ::= dato_tipo_int
 *   <año> ::= dato_tipo_int
 *   <mujer> ::= dato_tipo_bool
 *
 * Formato binario B02 para almacenar información de un grupo de personas
 * en un fichero:
 *   <fichero_B02> ::= { <persona> }
 *   <persona> ::= dato_tipo_persona
 */
```

```
/*
 * Pre: [nombre] define el nombre de un fichero que almacena información de
 *      un grupo de N personas según el formato textual T01.
 * Post: Ha asignado a numDatos el valor N y a tFechas[0,numDatos-1] y a
 *       tSexo[0,numDatos-1] la información correspondiente a cada una de
 *       las personas del grupo
 */
void leerDatosFormatoT01 (const char nombre[], int tFechas[], bool tSexo[],
                          int& numDatos);

/*
 * Pre: [nombre] define el nombre de un fichero que almacena información de
 *      un grupo de N personas según el formato textual T02.
 * Post: Ha asignado a numDatos el valor N y a tFechas[0,numDatos-1] y a
 *       tSexo[0,numDatos-1] la información correspondiente a cada una de las
 *       personas del grupo
 */
void leerDatosFormatoT02 (const char nombre[], int tFechas[], bool tSexo[],
                          int& numDatos);

/*
 * Pre: [nombre] define el nombre de un fichero que almacena información de
 *     un grupo de N personas según el formato binario B01.
 * Post: Ha asignado a numDatos el valor N y a tFechas[0,numDatos-1] y a
 *       tSexo[0,numDatos-1] la información correspondiente a cada una de las
 *       personas del grupo
 */
void leerDatosFormatoB01 (const char nombre[], int tFechas[], bool tSexo[],
                          int& numDatos);

/*
 * Pre: [nombre] define el nombre de un fichero que almacena información de
 *      un grupo de N personas según el formato binario B02.
 * Post: Ha asignado a numDatos el valor N y a tFechas[0,numDatos-1] y a
 *       tSexo[0,numDatos-1] la información correspondiente a cada una de las
 *       personas del grupo
 */
void leerDatosFormatoB02 (const char nombre[], int tFechas[], bool tSexo[],
                          int& numDatos);

/*
 * Pre: [nombre] define el nombre de un fichero destinado a almacenar información de
 *      un grupo de personas según el formato T01.
 * Post: Ha almacenado en el fichero [nombre] con formato textual T01 la información
 *       sobre un grupo de numPersonas personas definida en tFechas[0,numPersonas-1]
 *       y tSexo[0,numPersonas-1]
 */
void guardarDatosFormatoT01 (const char nombre[], int tFechas[], bool tSexo[],
                             int numPersonas);

/*
 * Pre: [nombre] define el nombre de un fichero destinado a almacenar información de
 *      personas según el formato textual T02.
 * Post: Ha almacenado en el fichero [nombre] con formato textual T02 la información sobre
 *       numPersonas personas definida en tFechas[0,numPersonas-1] y tSexo[0,numPersonas-1]
 */
void guardarDatosFormatoT02 (const char nombre[], int tFechas[], bool tSexo[],
                             int numPersonas);
```

```
/*
 * Pre: [nombre] define el nombre de un fichero destinado a almacenar información de
 *      personas según el formato binario B01.
 * Post: Ha almacenado en el fichero [nombre] con formato binario B01 la información sobre
 *       numPersonas personas definida en tFechas[0,numPersonas-1] y tSexo[0,numPersonas-1]
 */
void guardarDatosFormatoB01 (const char nombre[], int tFechas[], bool tSexo[],
                             int numPersonas);

/*
 * Pre: [nombre] define el nombre de un fichero destinado a almacenar información de
 *      personas según el formato binario B02.
 * Post: Ha almacenado en el fichero [nombre] con formato binario B02 la información sobre
 *       numPersonas personas definida en tFechas[0,numPersonas-1] y tSexo[0,numPersonas-1]
 */
void guardarDatosFormatoB02 (const char nombre[], int tFechas[], bool tSexo[],
                             int numPersonas);
```

El código del fichero de interfaz del módulo, el fichero `gestionFicheros.h`, se puede copiar de la sección de materiales docentes comunes de la web de la asignatura. Este fichero no debe ser alterado.

El fichero de implementación, `gestionFicheros.cc`, debe ser programado por cada alumno.

Para programar alguna de las funciones del módulo de biblioteca **gestionFicheros** será necesario haber programado previamente el módulo de biblioteca **persona** que se explica en el siguiente apartado.

5.2.3. Tercera tarea. Desarrollo de programas de prueba

Debe procederse a probar el buen comportamiento del código de cada una de las funciones de los módulo de biblioteca **gestionFicheros** y **persona**. Conviene probar cada una de estas funciones a medida que se van desarrollando, sin necesidad de esperar a que estén escritas las restantes.

Para ello se deben crear tantos proyectos diferentes (**Prueba01**, **Prueba02**, etc.) cuantos programas de prueba se vayan a desarrollar. Todos ellos se ubicarán en el área de trabajo **practica5**. Cada alumno tiene libertad para plantear los programas de prueba que considere más adecuados.

A título de ejemplo, se sugiere desarrollar como mínimo, los siguientes programas de prueba, apoyados en los recursos definidos en el módulo de biblioteca **grupo** programado en la práctica anterior, y en los módulos de biblioteca gestionFicheros y persona, programados en esta práctica.

- 1. Programa de prueba que mantenga el siguiente tipo de diálogo con el operador.

```
Nombre del fichero de personas: f01T01.txt
Formato del fichero anterior [T01,T02,B01,B02]: T01
Nombre del nuevo fichero de personas: nuevo01T01.txt
Se han copiado en el nuevo fichero los datos del ficheros de personas
ordenados de mayor a menor edad
```

El programa anterior lee los datos almacenados en un fichero de personas, almacenadas

según el formato seleccionado por el operador, los ordena cronológicamente y los guarda en un nuevo fichero bajo el mismo formato.

- 2. Programa de prueba que mantenga el siguiente tipo de diálogo con el operador.

```
Nombre del fichero de personas: f06B02.bin
Formato del fichero anterior [T01,T02,B01,B02]: B02
Nombre del nuevo fichero de personas: nuevo06B01.bin
Formato del fichero anterior [T01,T02,B01,B02]: B01
Se han copiado en el nuevo fichero los datos del fichero de personas
reorganizados de forma que los datos de mujeres precenden a los de hombres
```

  El programa anterior lee los datos almacenados en un fichero de personas, almacenadas según el primer formato seleccionado por el operador, los distribuye en función de su sexo y los guarda en un nuevo fichero bajo el segundo formato seleccionado.

Para facilitar la comprobación del correcto funcionamiento de los programas desarrollados en esta práctica se facilitan algunos ficheros de datos que pueden encontrase en la carpeta **datosPracticas/practica5** accesible desde la sección de materiales docentes comunes de la web de la asignatura.

Práctica 6: Definición de nuevos tipos de datos e implementación de una colección de funciones básicas para trabajar con ellos

6.1. Objetivos de la práctica

En esta práctica se debe decidir cómo representar un nuevo tipo de dato y, a continuación, programar una colección de funciones básicas para trabajar con datos de dicho tipo.

El nuevo tipo, denominado **ListaPersonas**, facilitará la gestión de una lista de datos del tipo **Persona**, diseñado en la práctica anterior.

Para ello se diseñará un módulo de biblioteca denominado **listaPersonas** que encapsule la representación del nuevo tipo y el código de las funciones básicas citadas.

Desde otros módulos se podrá hacer uso del tipo **ListaPersonas** para definir datos de ese nuevo tipo, así como de las funciones especificadas en la interfaz del módulo **ListaPersonas**.

Por el contrario, desde otros módulos no se podrá hacer uso de los detalles sobre cómo ha sido definido el tipo **ListaPersonas** por una poderosa razón metodológica: para permitir posibles cambios que mejoren la definición del tipo **ListaPersonas** sin necesidad de modificar ni una línea de código de los módulos que hagan uso del tipo **ListaPersonas**.

La práctica continúa con el diseño del módulo de biblioteca **funciones** que presenta una pequeña colección de funciones que trabajan con listas de personas representadas mediante un dato de tipo **listaPersonas**. La práctica concluye con el diseño de los programas de prueba que cada alumno considere conveniente desarrollar.

Al igual que en las restantes prácticas, a la sesión de laboratorio asociada a esta práctica hay que acudir con el trabajo que se describe a continuación ya hecho y aprovecharla con un doble objetivo:

- Consultar al profesor allí presente las dudas surgidas y las dificultades que hayan impedido resolver satisfactoriamente los problemas planteados y, con su ayuda, tratar de aclarar ideas y avanzar en la resolución de los problemas surgidos.
- Presentar al profesor el trabajo realizado para que este lo pueda revisar, advertir de posibles errores y defectos y dar indicaciones sobre cómo mejorarlo y, en su caso, corregirlo.

6.2. Arquitectura modular de los programas a desarrollar

Los programas a desarrollar alrededor de esta práctica tienen una estructura modular. Los módulos que la integran se describen a continuación.

6.2.1. El módulo de biblioteca `persona`

Este módulo de biblioteca fue desarrollado en la práctica anterior.

6.2.2. El módulo de biblioteca `listaPersonas`

El módulo de biblioteca **listaPersonas** debe ofrecer a los restantes módulos:

- La definición del tipo de dato **ListaPersonas**, que permite representar la información de una lista de datos de tipo **Persona** (véase la práctica 5ª) con capacidad para **DIM** personas, donde **DIM** es una constante definida en el propio módulo.
- Una colección de funciones para trabajar con datos de tipo **ListaPersonas**: `nuevaLista()`, `numPersonas(L)`, `consultar(L,i)`, `anyadir(L,p)`, `retirar(L)`, `insertar(L,p,i)` y `eliminar(L,i)`.

Un listado parcial del fichero de interfaz del módulo **listaPersonas** se presenta a continuación (falta únicamente definir la estructura interna del tipo **ListaPersonas**). Este listado parcial se puede encontrar accediendo al enlace de materiales docentes comunes de la web de la asignatura.

```
/*
 * Fichero de interfaz listaPersonas.h del módulo listaPersonas
 */

#ifndef LISTA_PERSONAS_H

#define LISTA_PERSONAS_H

#include "../../practica5/Persona/persona.h"

const int DIM = 200; // Capacidad máxima de las listas de personas a definir

/*
 * Representa una lista de datos de tipo Persona con capacidad
 * máxima de almacenamiento igual a DIM
 */
struct ListaPersonas {
    /*
     * La estructura interna de este tipo debe ser definida por cada alumno
     */
     . . .
     . . .
};

/*
 * Pre: ---
```

```
 * Post: Devuelve una lista que no almacena ninguna persona, es decir,
 *       devuelve la lista <>
 */
ListaPersonas nuevaLista ();

/*
 * Pre: La lista L almacena K personas, es decir, L = < p_1, p_2, ..., p_K >
 * Post: Devuelve el número K de personas almacenadas en la lista L
 */
int numPersonas (const ListaPersonas L);

/*
 * Pre: La lista L almacena K personas, es decir, L = < p_1, p_2, ..., p_K >
 *      y i >= 1 y i <= K
 * Post: Devuelve p_i, es decir, la persona ubicada en la posición i
 *       de la lista L
 */
Persona consultar (const ListaPersonas L, const int i);

/*
 * Pre: La lista L almacena K personas, es decir, L = < p_1, p_2, ..., p_K >
 *      y K < DIM
 * Post: La lista L almacena K+1 personas ya que a las K personas que
 *       almacenaba inicialmente se ha incorporado p como último elemento de la
 *       lista, es decir, ahora L = < p_1, p_2, ..., p_K, p >
 */
void anyadir (ListaPersonas& L, const Persona p);

/*
 * Pre: La lista L almacena K personas, es decir, L = {p_1, p_2, ..., p_K}
 *      y K > 0
 * Post: Devuelve la persona que ocupaba el primer lugar de la lista L, es
 *       decir, p_1, y modifica la lista L eliminando de ella la persona p_1,
 *       es decir, ahora L = {p_2, ..., p_K}
 */
Persona retirar (ListaPersonas& L);

/*
 * Pre: La lista L, siendo L = < p_1, p2, ..., p_i-1, p_i ,..., p_K >, almacena K
 *      personas, K < DIM, i >= 1 e i <= K
 * Post: La lista L almacena K+1 personas ya que a las K personas que almacenaba
 *       inicialmente se ha incorporado [p] como i-ésimo elemento de la lista, es decir,
 *       ahora L = < p_1, p2, ..., p_i-1, p, p_i, ..., p_K >
 */
void insertar (ListaPersonas& L, const Persona p, const int i);

/*
 * Pre: La lista L, siendo, L = < p_1, p2, ..., p_i-1, p_i, p_i+1, ..., p_K >,
 *      almacena K personas, K > 0, i >= 1 e i < = K
 * Post: Modifica la lista L eliminando de ella la persona p_i, es decir ahora solo
 *       almacena K-1 personas, siendo L = < p_1, p2, ..., p_i-1, p_i+1, ..., p_K >
 */
void eliminar (ListaPersonas& L, const int i);

#endif
```

6.2.3. El módulo de biblioteca `funciones`

El módulo de biblioteca **funciones** debe ofrecer a los restantes módulos las siguientes funciones:

- Función `mostrar(p)` que permite mostrar por pantalla la información de la persona `p`.
- Función `mostrar(L)` que permite mostrar por pantalla la información de la lista de personas `l`.
- Función `masJoven(L)` que devuelve el dato de tipo **Persona** correspondiente a la persona más joven de las incluida en la lista `L`.
- Función `ordenar(L)` que reorganiza los datos de la lista `L` de forma que finalmente quedan ordenados según edades decrecientes.

Un listado completo del fichero de interfaz del módulo **funciones** se presenta a continuación. Este listado se puede encontrar accediendo al enlace de materiales docentes comunes de la web de la asignatura.

```
/*
 *  Fichero de interfaz funciones.h del módulo funciones
 */

#ifndef FUNCIONES_H

#define FUNCIONES_H

 #include "../Lista/listaPersonas.h"

using namespace std;

/*
 * Pre: --
 * Post: Ha presentado por pantalla, completando la línea en curso, la información
 *       de la persona p con un formato idéntico al que se muestra a continuación:
 *       07/06/1974 hombre
 */
void mostrar (const Persona p);

/*
 * Pre: L = < p_1, p_2, ..., p_K > y K >= 0
 * Post: Ha presentado un listado por pantalla con la información de las K personas
 *       de la lista L, en el orden que están en la lista, a razón de una persona
 *       por línea. Presenta la información con un formato idéntico al que se muestra
 *       a continuación:
 *           1. 31/12/1982 mujer
 *           2. 01/01/1982 hombre
 *           3. 17/03/2004 hombre
 *                 . . .
 *         186. 09/04/1999 mujer
 *         187. 19/10/1983 hombre
 */
void mostrar (const ListaPersonas L);
```

```
/*
 * Pre: L = < p_1, ..., p_K > y K > 0
 * Post: Devuelve la persona de la lista L cuya fecha de
 *       nacimiento es posterior
 */
Persona masJoven (const ListaPersonas L);

/*
 * Pre: L = < p_1, ..., p_K > y K >= 0
 * Post: Ha permutado las personas de la lista L de forma que
 *       ahora están ordenada por fecha de nacimiento, comenzando
 *       por las de más edad y terminando por las más jóvenes
 */
void ordenar (ListaPersonas& L);

#endif
```

6.2.4. Programas de prueba

Cada alumno es responsable de poner a punto cuantos programas de prueba sean necesarios para verificar el buen comportamiento de los recursos programados en los módulos de biblioteca **listaPersonas** y **funciones**.

Cada uno de estos programas tendrá una arquitectura modular integrada por cuatro niveles;

- Primer nivel o nivel superior de abstracción. En este nivel se sitúa el módulo principal de cada uno de los programas de prueba cuyo diseño se apoya en los tres niveles que siguen.
- Segundo nivel de abstracción. En este nivel se sitúa el módulo **funciones** cuyas funciones han de ser programadas haciendo uso de recursos definidos en los dos niveles que siguen.
- Tercer nivel de abstracción. En este nivel se sitúa el módulo **listaPersonas** en cuya definición del tipo **ListaPersonas** y del código de sus funciones se ha de hacer uso de recursos definidos en el nivel que sigue.
- Cuarto nivel o nivel inferior de abstracción. En este nivel se sitúa el módulo **Persona** programado en la práctica anterior.

Es esencial, desde un punto de vista metodológico, que el código desarrollado al programar cualquiera de los módulos descritos no debiera tener que ser modificado si en alguno de los módulos de menor nivel de abstracción se decide cambiar la definición interna de alguno de los tipos de datos, es decir, la definición del tipo **Persona** en el módulo **persona** o la definición del tipo **ListaPersonas** en el módulo **ListaPersonas**.

6.3. Trabajo a desarrollar

Cada alumno ha de desarrollar en el área de trabajo **practica6** ubicada en la carpeta **practicasPROG1** los proyectos que se describen a continuación.

- Proyecto **Lista**. En este proyecto se desarrollará únicamente el módulo **listaPersonas** con sus ficheros de interfaz y de implementación `listaPersonas.h` y `listaPersonas.cc`, respectivamente.

- Proyecto **FuncionesLista**. En este proyecto se desarrollará únicamente el módulo **funciones** con sus ficheros de interfaz y de implementación `funciones.h` y `funciones.cc`, respectivamente.
- Proyectos **Pruebas01**, **Pruebas02**, **Pruebas03**, etc.. En cada uno de estos proyectos se desarrollará un programa de prueba, según se ha indicado en el apartado enterior.