Práctica 3: Diseño modular de programas escritos en C++ que trabajan con datos numéricos

3.1. Objetivos de la práctica

Los objetivos de la práctica son los siguientes:

- Desarrollar algunos módulos de biblioteca en los que se definen funciones que operan con datos naturales y enteros.
- Programar algunas funciones que resuelven problemas de tratamiento de datos numéricos naturales y enteros.
- Aprender a poner a punto programas modulares en C++, desarrollando módulos de biblioteca, por un lado, y programas C++ que hacen uso de ellos, por otro.

Al igual que en la práctica anterior, lo deseable es que cada alumno acuda a la sesión asociada a la práctica con el trabajo ya hecho y aproveche la sesión de prácticas para resolver dudas, completar aquellas tareas en la que hayan surgido dificultades no superadas y para que el profesor supervise el trabajo realizado y pueda hacer sugerencias para su mejora.

3.2. Tecnología y herramientas

Los proyectos de programación a desarrollar en esta práctica se localizarán en una nueva área de trabajo denominada «Práctica 3» que ha de ser guardada en el directorio `Practica3` que deberá haber sido creado en el directorio `Programacion1`.

Parte del código fuente correspondiente a esta práctica va a ser almacenado en un directorio distinto a `Practica3`. En concreto, los ficheros de código fuente de los módulos de biblioteca que se van a desarrollar en esta práctica, van a estar ubicados en el directorio `Biblioteca` del directorio `Programacion1`. Con esto, se pretende facilitar la reutilización del código correspondiente a dichos módulos en prácticas posteriores.

3.2.1. Desarrollo y puesta a punto de un programa modular

Estructura modular del programa

Todo programa C++ consta de un módulo principal que incluye la función `main` o función principal del programa. Un programa C++ puede presentar una estructura modular si, además de un módulo principal, consta de uno o más módulos de biblioteca adicionales.

Se va a poner a punto el programa interactivo y dirigido por menú de estructura modular presentado en el tema 7 de la asignatura. El programa consta de un módulo principal y de un módulo de biblioteca, el módulo `enteros`. Los tres ficheros, el que contiene el código del módulo principal y los correspondientes a los ficheros de interfaz y de implementación del módulo de biblioteca `enteros`, se puede copiar accediendo a su código a través de la carpeta `Practica3` de la sección «Código C++ y datos»de la web de esta asignatura.

- Fichero `calculadora-enteros.cpp` con el código del módulo principal del programa.
- Fichero `enteros.h` de interfaz del módulo `enteros`.
- Fichero `enteros.cpp` de implementación del módulo `enteros`.

Uso de la directiva `#include`

Los recursos definidos en la interfaz del módulo de biblioteca `enteros` han de ser visibles desde el módulo principal. Para lograrlo, en el código del módulo principal, localizado en el fichero `calculadora-enteros.cpp`, se ha de escribir una directiva **`#include`** que inserta en el código a compilar el código del fichero de interfaz del módulo `enteros`.

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

Esta directiva hace que el preprocesador de C++ trate el contenido del fichero de interfaz especificado (`calculos.h`) como si estuviera escrito en el propio fichero `calculadora-enteros.h`, en el punto donde se ha situado la directiva.

Con ello se logra dar visibilidad a los elementos declarados en el fichero de interfaz del módulo de biblioteca `enteros` a partir de ese punto del módulo principal del programa.

Conviene insistir que el fichero que hay que especificar en la directiva **`#include`** es el fichero de interfaz del módulo (el de extensión `.h`) y no el de implementación.

Cuando se presentó el programa en clase de teoría, la directiva **`#include`** que se puso en el fichero `calculadora-enteros.cpp` era **`#include`** `"calculos.h"`, lo que suponía que el fichero `calculos.h` estaba almacenado en el mismo directorio que el fichero `calculadora-enteros.cpp`.

Como se ha comentado anteriormente, en esta práctica y las siguientes, con el objeto de facilitar la reutilización de las funciones definidas en los módulos de de biblioteca, se van a almacenar los ficheros con el código fuente correspondiente en un directorio específico, denominado `Biblioteca` y que se encuentra dentro del directorio `Programacion1`, al mismo nivel que el resto de los directorios de las prácticas (figura 3.2.1).

Por ello, en el **`#include`** del fichero `calculadora-enteros.cpp` no solo hay que poner el nombre del fichero de interfaz `calculos.h`, sino una *ruta de acceso* al mismo. Dicha ruta podría ser *absoluta* (por ejemplo `/home/a123456/Programacion1/Biblioteca/calculos.h`), pero probablemente requeriría ser modificada cada vez que se moviera el código de un equipo a otro. Otra posibilidad mu-

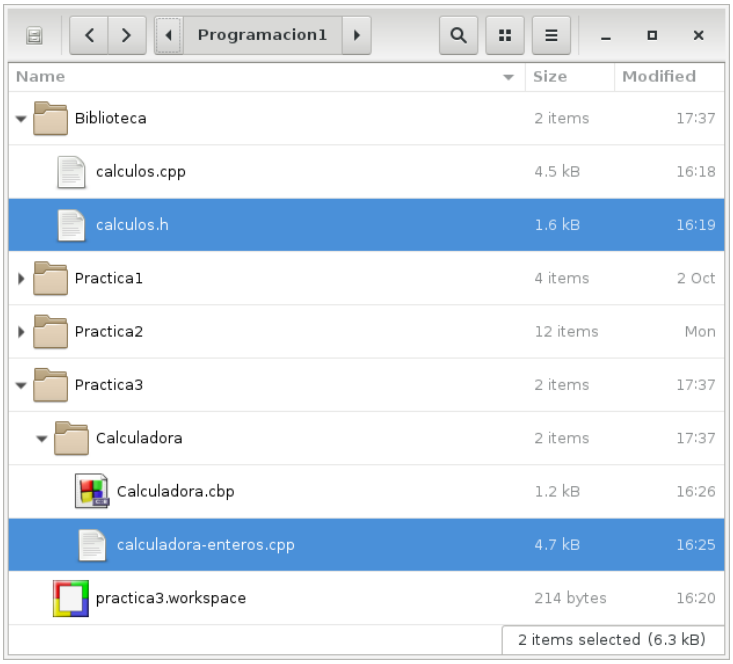

Figura 3.1: Contenido del directorio `Programacion1`, donde se han seleccionado el fichero de interfaz `calculos.h`, dentro del directorio `Biblioteca`, y el fichero del módulo principal, `calculadora-enteros.cpp`, que se encuentra en el directorio `Practica3/Calculadora`.

cho más adecuada es escribir una ruta a `calculos.h` *relativa* al directorio en el que se encuentra `calculadora-enteros.cpp`. Este se halla en el directorio `Calculadora` del directorio `Practica3` del directorio `Programacion1`.

En términos generales, la ruta `..` relativa a un determinado directorio hace referencia al directorio que lo contiene o *directorio padre*. Así, la ruta `..` relativa al directorio `Calculadora` es el directorio `Practica3`. La ruta `../..` relativa a `Calculadora` hace referencia al directorio `Programacion1`. Del mismo modo, la ruta `../../Biblioteca` hace referencia al directorio `Programacion1/Biblioteca`. Es por ello que el nombre del fichero que se pone en la directiva **`#include`** del fichero `calculadora-enteros.cpp` es `../../Biblioteca/calculos.h`.

Uso de las directivsa `#ifndef` y `#endif`

En un programa que conste de varios módulos, pudiera darse el caso de que un mismo fichero de interfaz se incluyera más de una vez a través de varias directivas **`#include`**. Ello podría provocar errores de compilación al estar replicadas las declaraciones y definiciones de los elementos declarados en la interfaz de dicho módulo. Para evitar esos posibles errores, se hace uso en el fichero de interfaz del módulo de las directivas **`#define`**, **`#ifndef`** y **`#endif`**, dirigidas al preprocesador de C++. Los detalles sobre el uso de las tres directivas citadas se explican a continuación:

- La directiva **`#define`** cuyo efecto es definir una *macro*, es decir, un identificador que será utilizado posteriormente por el preprocesador de C++. Así, por ejemplo, la siguiente directiva escrita al comienzo del fichero de interfaz del módulo `propiedades-enteros` define la macro `CALCULOS_H_INCLUDED`. El nombre de esta macro es independiente del código C++ del programa y solo es relevante para el preprocesador de C++.

```
#define CALCULOS_H_INCLUDED
```

- Una par de directivas **`#ifndef`** *`macro`* ... **`#endif`** que indican al preprocesador de C++ que el código situado entre su comienzo, **`#ifndef`** *`macro`*, y su final, **`#endif`**, solo debe ser tenido en cuenta si *`macro`* no ha sido definida previamente.

De esta forma se logra que el contendido del fichero de interfaz de un módulo sea incluido como mucho una sola vez por el preprocesador de C++ en el código del programa a compilar, independientemente del número de cláusulas `\#include` figure el nombre de dicho fichero de interfaz.

Creación y configuración del proyecto «`Calculadora`»

1. En primer lugar, el área de trabajo por defecto se va a renombrar a «Práctica 3» y se va a guardar en el directorio `Programacion1/Practica3`, de forma equivalente a como se ha hecho en prácticas anteriores.

2. Debe crearse un proyecto denominado «`Calculadora`» en el directorio `Practica3` correspondiente al área de trabajo «Práctica 3». En él debe ubicarse el código del módulo principal del programa, el fichero `calculadora-enteros.cpp`.

3. Los ficheros de interfaz e implementación del módulo `calculos` (es decir, los ficheros `enteros.h` y `enteros.cpp`) deben ser ubicardos en el directorio `Programacion1/Biblioteca`, ya que desde el módulo principal se hace uso de recursos definidos en el módulo `enteros` y es el directorio `Biblioteca` donde espera encontrarlos. Por esta razón, el fichero `enteros.h` debe ser visible desde el módulo principal del programa y el fichero `enteros.cpp` debe añadirse al proyecto para que su código sea compilado y utilizado al generar el fichero ejecutable.

   Para ello existen dos alternativas:

   a) La forma más simple de crear nuevos ficheros desde Code::Blocks y añadirlos a un proyecto es ejecutar la orden *«File»* → *«New»* → *«File...»* y seleccionar la plantilla (*template*) *«C/C++ header»* o *«C/C++ source»*, según se trate de crear el fichero de interfaz o el de implementación del módulo, respectivamente.

   El asistente que aparece a continuación solicita el nombre y ruta completos del fichero que deseamos crear. Se puede utilizar el botón que abre el explorador de ficheros (botón *«...»*) para buscar el directorio `Programacion1/Biblioteca` (creando el directorio `Biblioteca` si es preciso) y escribir allí el nombre del fichero. Al final, en el cuadro de texto *«Filename with full path:»* debe aparecer una ruta como `/home/a123456/Programacion1/Biblioteca/calculos.h`, suponiendo que se está creando el fichero `calculos.h` y donde el nombre de usuario en `hendrix` fuese `a123456`. En otros sistemas operativos, la parte inicial de la ruta puede variar, pero en cualquier caso debería acabar en `Programacion1/Biblioteca/calculos.h` o `Programacion1\Biblioteca\calculos.h`.

   Al crear ficheros de implementación de un módulo, hay que asegurarse de incluirlo en los modos de compilación *«Debug»* y *«Release»*, marcando ambas casillas.

   Una vez creado, puede copiarse su código desde la web de la asignatura y pegarse en el fichero a través de Code::Blocks.

   b) La forma más cómoda es descargar los ficheros directamente de la web de la asignatura y añadirlos luego al proyecto en Code::Blocks.

   Se pueden descargar los ficheros en las páginas de la web que muestran listados de ficheros. En lugar de hacer clic en un fichero, debe hacerse clic con el botón secundario del ratón y elegir la opción *«Guardar destino como»*, *«Guardar enlace como...»* o similar, dependiendo el navegador utilizado.

En el cuadro diálogo que se abre, hay que buscar el directorio `Programacion1/Biblioteca` (creando el directorio `Biblioteca` si es preciso), para guardar allí el fichero que estemos descargando.

A continuación, en Code::Blocks, hay que seleccionar el proyecto con el ratón, pulsar su botón derecho y ejecutar la orden *«Add files...»*, para seleccionar a continuación los ficheros `enteros.h` y `enteros.cpp` ubicados en el directorio `Programacion1/Biblioteca`.

Al añadirlos, Code::Blocks preguntará por las configuraciones de compilación a las que añadirlos. Por defecto ya estarán seleccionados los modos *«Debug»* y *«Release»*, y se dejará dicha selección así.

4. Una vez realizadas las tareas anteriores puede procederse a compilar el código del proyecto «`Calculadora`» y construir un programa ejecutable. Si no hay errores de compilación (debidos a una configuración incorrecta del proyecto o a una ubicación de algún fichero en un directorio distinto al previsto), se podrá ejecutar el programa resultante y comprobar su correcto comportamiento.

Aquí finaliza el material que podrá utilizar el profesor para ilustrar, al comienzo de la sesión, el modo de poner a punto programas modulares escritos en C++ con la ayuda del entorno Code::Blocks. En el siguiente apartado se describe el trabajo que se propone que los alumnos desarrollen de forma autónoma.

3.3. Trabajo a desarrollar en esta práctica

Cada alumno debe completar las tareas que se describen a continuación, realizando un trabajo suficiente con anterioridad a la sesión de prácticas que le corresponda, con el objeto de sacar el máximo rendimiento de dicha sesión.

3.3.1. Primera tarea

Crea y configura el proyecto «`Calculadora`», tal y como se ha explicado en la sección anterior. Comprueba que compila sin errores y que el programa resultante puede ejecutarse y funciona correctamente.

3.3.2. Segunda tarea

En el fichero de interfaz del módulo de biblioteca `calculos` (fichero `calculos.h`), hay dos funciones declaradas (`fibonacci` y `mcm`) cuya implementación no se encuentra en el fichero de implementación `calculos.cpp`. Esto no ha supuesto ningún problema hasta el momento, ya que el módulo principal no estaba haciendo uso de ellas.

Escribe en el fichero de implementación `calculos.cpp` el código de las funciones `fibonacci` y `mcm`. Este código debe ser diseñado por cada uno de vosotros. A modo de referencia, se presenta a continuación la especificación completa de las dos funciones, tal y como ya aparecen en el fichero de interfaz del módulo.

```
/*
 *  Pre: a != 0 o b != 0
 *  Post: Ha devuelto el mímimo común múltiplo de «a» y «b».
 */
```

```
int mcm(int a, int b);


/*
 *  Pre:  n >= 1
 *  Post: Ha devuelto el n-ésimo término de la sucesión de Fibonnaci.
 *  Nota: La sucesión de Fibonacci es una sucesión infinita de números
 *        naturales que comienza con los números 0 y 1 y cuyos restantes
 *        términos son iguales a la suma de los dos que le preceden. Estos
 *        son los primeros términos de esta sucesión infinita:
 *          0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, ...
 */
int fibonacci(int n);
```

3.3.3. Tercera tarea

El trabajo de desarrollo de un programa no concluye hasta que se han realizado las pruebas necesarias para validar su buen comportamiento.

Las funciones del módulo `calculos` pueden probarse indirectamente a través de programas como, por ejemplo, el desarrollado en el proyecto «`Calculadora`», que puede servir para comprobar el buen funcionamiento de las funciones `numCifras`, `sumaCifras`, `cifra`, `imagen` y `esPrimo` del módulo `calculos`. Sin embargo, también es posible hacer pruebas directamente sobre las propias funciones a través de un programa o programas específicos que realicen invocaciones a las funciones y comprueben los resultados obtenidos.

Crea y configura un proyecto denominado «`CalculadoraTest`» en el directorio `Practica3` del área de trabajo «Práctica 3». Incluye en dicho proyecto los ficheros `calculos-test.h`, `calculos-test.cpp` y `calculos-test-main.cpp` que puedes descargar de la página web de la asignatura. Agrega también al proyecto los ficheros `calculos.h` y `calculos.cpp` que ya tienes en el directorio `Biblioteca`.

Si lo has hecho correctamente, el proyecto «`CalculadoraTest`» compilará sin errores y, al ser ejecutado, realizará varias invocaciones a las funciones `fibonacci` y `mcm` e informará de si el resultado de la ejecución es correcto o no. Hecha un vistazo al código de las funciones que hacen las pruebas. Si, al ejecutar el programa, este te informa de que alguna invocación no devuelve los resultados esperados, corrige la función de que se trate.

El programa correspondiente a este poryecto permite validar parcialmente el trabajo que hayas desarrollado en el módulo `calculos` con respecto a las funciones `fibonacci` y `mcm` y, en su caso, detectar y corregir errores. En todo caso, que el programa de pruebas no detecte errores no es garantía de que no los haya.

3.3.4. Cuarta tarea

Añade al programa correspondiente al proyecto «`Calculadora`» dos opciones de menú adicionales a las ya existentes que permitan al usuario calcular el mínimo común múltiplo de dos números, por una parte y obtener el valor de un término de la sucesión de Fibonacci elegido por él mismo, por otra. Añade el código necesario para que dichas opciones de menú se ejecuten cuando sean elegidas por el usuario, apoyándote, por supuesto, en las funciones que has implementado en el módulo `calculos`.

3.3.5. Quinta tarea

La quinta y sexta tarea de esta práctica se añadirán a lo largo de la semana del 22 al 26 de noviembre.