Fundamentos de Programación.

Guión de Prácticas.

Curso 2012/2013

Autor: Juan Carlos Cubero
Para cualquier sugerencia o comentario sobre este guión de prácticas, por favor, enviad un e-mail a JC.Cubero@decsai.ugr.es

Profesores de prácticas: M.Gómez y S. Acid
Para cualquier cuestión metodológica ponerse en contacto con los profesores o bien enviando un e-mail a mgomez@decsai.ugr.esy acid@decsai.ugr.es

"Lo que tenemos que aprender a hacer, lo aprendemos haciéndolo".
Aristóteles

"In theory, there is no difference between theory and practice. But, in practice, there is".
Jan L. A. van de Snepscheut

"The gap between theory and practice is not as wide in theory as it is in practice".

"Theory is when you know something, but it doesn't work. Practice is when something works, but you don't know why. Programmers combine theory and practice: Nothing works and they don't know why".

Sobre el guión de prácticas

Este guión de prácticas contiene las actividades a realizar por el alumno, tanto de forma presencial (en las aulas de la Escuela de Informática) como no presencial a realizar en casa. Todas las actividades son obligatorias y están incluidas en la evaluación de la participacion de clase, excepto las marcadas como *Actividades de Ampliación*.

El guión está dividido en sesiones, que corresponden a cada semana lectiva. Como ya se indicara en clase, para la asignatura de FP a cada semana le corresponden 6 horas de trabajo personal del alumno, aparte de las 4 horas de clases presenciales. Por ello, las actividades no presenciales de una sesión deben desarrollarse **antes**, *durante toda una semana*, y culminan con una entrega de los ejercicios en la propia sesión de prácticas. Por lo que el alumno deberá descargarse al terminar una sesión, las actividades encargadas para la sesión siguiente.

Entre las actividades del guión, una actividad no presencial recurrente es, la resolución de problemas propuestos en las *Relaciones de Problemas*. Éstas constan de dos tipos de problemas:

- *Básicos*:

  Deben resolverse en la casa y el alumno deberá defenderlos individualmente. La defensa será individual y forma parte de la nota final de la asignatura (10 % tal y como se explica en el apartado de evaluación de la asignatura).

  Las soluciones a los problemas deberán ser subidas a la plataforma de decsai através del GAP. Para ello, el alumno debe entrar en el acceso identificado de decsai, entrar por el GAP, y seleccionar la entrega de prácticas correspondiente a la semana en curso. El alumno subirá cada uno de los ficheros con extensión `cpp` correspondientes a las soluciones de los ejercicios porpuestos (tanto obligatorios, como opcionales) fijados en esa sesión.
  Nota: Para una correcta gestión de los ficheros en el GAP, los nombres de los ficheros deberán ser de la forma: `solucion1.cpp`, `solucion2.cpp`, `solucion3.cpp` etc. correspondientes al número de ejercicio con el que figuran en el guión de prácticas.

  El profesor entregará al final de cada semana las soluciones a los ejercicios básicos. Es muy importante que el alumno revise estas soluciones y las compare con las que él había diseñado.

  Del conjunto de problemas básicos, en el guión de prácticas se distinguirá entre:

  1. *Obligatorios*:

     Si se realizan correctamente estos ejercicios, el alumno podrá sacar hasta un 9 (sobre 10) de nota.

2. *Opcionales*:
   Si se realizan correctamente estos ejercicios, el alumno podrá sacar hasta un 10 (sobre 10) de nota.

- *Ampliación*: problemas cuya solución no se verá, pero que sirven para afianzar conocimientos. El alumno debería intentar resolver por su cuenta un alto porcentaje de éstos.

Las actividades marcadas como *Seminario* han de hacerse obligatoriamente y siempre serán expuestas por algún alumno elegido aleatoriamente.

Para la realización de estas prácticas, se utilizará el entorno de programación `Code::Blocks` Se recomienda la instalación del programa lo antes posible (en la primera semana de prácticas). En cualquier caso, el alumno puede instalar en su casa cualquier otro compilador, como por ejemplo `Visual Studio` → `www.microsoft.com/visualstudio/`

Muy importante:

- La resolución de los problemas y actividades puede hacerse en grupo, pero la defensa durante las sesiones presenciales es individual.
- LLevar la asignatura al día para poder realizar los ejercicios propuestos para cada sesión.

RELACIÓN DE PROBLEMAS III. Funciones y Clases

Problemas Básicos

Problemas sobre funciones

1. Cread una función que calcule el máximo entre tres `double`. La cabecera de la función será la siguiente:

   ```
   double Max(double un_valor, double otro_valor, double el_tercero)
   ```

   Construid un programa principal que llame a dicha función con los parámetros actuales que se quiera.

   Finalidad: Familiarizarnos con la definición de funciones, el paso de parámetros y el ámbito de las variables. Dificultad Baja.

2. Cread una función para calcular la suma de una progresión geométrica, tal y como se vio en el ejercicio 19 de la relación de problemas II. Analizad cuáles deben ser los parámetros de dicha función.

   Finalidad: Diseño de una función sencilla. Dificultad Baja.

3. En el ejercicio 7 de la relación de problemas II (sobre los descuentos a aplicar a una tarifa aérea), se repetía en varias ocasiones un código muy parecido, a saber:

   ```
   rebaja = rebaja + tarifa_con_km_adicionales *
                     DESCUENTO_CLIENTE_PREVIO/100.0;
   rebaja = rebaja + tarifa_con_km_adicionales *
                     DESCUENTO_TRAYECTO_MEDIO/100.0;
   ...........
   ```

   Construid la función apropiada para que no se repita código y re-escribid la solución llamando a dicha función en todos los sitios donde sea necesario.

   Finalidad: Diseño de una función sencilla. Dificultad Baja.

4. Se pide definir las siguientes funciones y cread un programa principal de ejemplo que las llame:

   a) `EsMinuscula` comprueba si un carácter pasado como parámetro es una minúscula. A dicho parámetro, llamadlo `una_letra`.

b) `Convierte_a_Mayuscula` comprueba si un carácter pasado como parámetro es minúscula (para ello, debe llamar a la función anterior), en cuyo caso lo transforma a mayúscula. Si el carácter no es minúscula debe dejar la letra igual. A dicho parámetro, llamadlo `caracter`.
Esta función hace lo mismo que la función `tolower` de la biblioteca `cctype`

Observad que el parámetro `una_letra` de la función `EsMinuscula` podría llamarse igual que el parámetro `caracter` de la otra función. Esto es porque están en ámbitos distintos y para el compilador son dos variables distintas. Haced el cambio y comprobarlo.

Finalidad: Entender el ámbito de los parámetros de las funciones, así como las llamadas entre funciones. Dificultad Baja.

5. En el ejercicio 5 de la relación de problemas I (página 4) se vio cómo obtener el valor de ordenada asignado por la función gaussiana, sabiendo el valor de abscisa x. Recordemos que esta función matemática dependía de dos parámetros μ (esperanza) y σ (desviación), y venía definida por:

$$\text{gaussiana}(x) = \frac{1}{\sigma\sqrt{2\pi}} \quad e^{\left\{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right\}}$$

Cread una función para calcular el valor de la función gaussiana en un punto arbitrario leído desde teclado. Cread un programa principal que llame a la función.

Finalidad: Entender la separación de E/S con Cómputos en el contexto de las funciones. Dificultad Baja.

6. Vamos a añadir una función al ejercicio 4 de esta relación de problemas. Se pide implementar la función `Transforma`, a la que se le pase como parámetro un `char` y haga lo siguiente:

   - si el argumento es una letra en mayúscula, devuelve su correspondiente letra en minúscula,
   - si el argumento es una letra en minúscula, devuelve su correspondiente letra en mayúscula,
   - si el argumento no es ni una letra mayúscula, ni una letra mayúscula, devuelve el carácter pasado como argumento.

   Observad que la constante de amplitud

   ```
   const int AMPLITUD = 'a'-'A';
   ```

   es necesaria declararla como constante local en ambas funciones. Para no repetir este código, ¿qué podemos hacer?

   Finalidad: Entender cómo se llaman las funciones entre sí. Dificultad Baja.

Problemas sobre clases

7. Se pide crear una clase `Conjunto_3_Valores` que represente un conjunto de 3 valores de tipo `double`, sobre los que se quiere realizar las operaciones de calcular el máximo y la media aritmética. Para ello, haced lo siguiente:

   a) Definid en primer lugar la clase con tres datos miembro públicos llamados `uno`, `dos` y `tres`. A esta primera versión de la clase, dadle el nombre `Conjunto_3_Valores_vs1`

   Añadid un método público `Max`. Este método tendrá la siguiente cabecera:

   ```
   double Max()
   ```

   Observad que el método `Max` actúa sobre los datos de la clase (`uno`, `dos` y `tres`) por lo que no tiene parámetros. Observad la diferencia con la cabecera del ejercicio 1 en el que `Max` era una función que no estaba dentro de ninguna clase.

   b) Añadid ahora un método para calcular la media aritmética de los tres valores.

   c) Cread un `main` que llame a los métodos anteriores.

   Comprobad que podemos modificar directamente los datos miembro públicos desde el `main`, lo cual no es muy recomendable como ya se ha visto en clase de teoría. Para resolver este problema, hacemos lo siguiente:

   d) Copiad el código de la clase anterior y copiadlo dentro de una nueva clase llamada `Conjunto_3_Valores_vs2`

   En esta nueva versión, declarad los datos miembro como `private`, en vez de `public`. Comprobad que ya no podemos acceder a ellos desde el `main`, por lo que necesitamos asignarles un valor de otra forma. Para ello, haced lo siguiente:

      I) Añadid un constructor a la clase `Conjunto_3_Valores_vs2`. En dicho constructor, pasaremos los tres valores que habrá que asignárselos a los datos miembro (usad la lista de inicialización, que va precedida de dos puntos).
      Cread el objeto correspondiente desde el `main` (el compilador nos obligará a pasar los tres valores en el constructor)

      II) Añadid tres métodos (`GetPrimero()`, `GetSegundo()`, `GetTercero()`) que devuelvan los correspondientes datos miembro, para que así pueda consultarse su valor desde el `main`.

   e) Supongamos que creamos un objeto con los valores `2, 5, 7`. Si ahora quisiéramos cambiar de conjunto de datos y trabajar con `3, 9, 1`, por ejemplo, tendríamos que crear otro objeto distinto. Esto es completamente lícito. En cualquier

caso, dependiendo del problema que estemos abordando, podríamos estar interesados en permitir cambiar los datos del objeto (su *estado*) y así no tener que crear otro objeto distinto. Para ello, debemos proporcionar dicha funcionalidad a la clase. Lo hacemos de tres formas alternativas:

- Creamos la clase
  `Conjunto_3_Valores_vs3`
  que contendrá el mismo código que `Conjunto_3_Valores_vs2` y le añadimos un método `SetValores` que cambie los tres valores a la vez (como ha cambiado el nombre de la clase, habrá que cambiar el nombre del constructor). Cread un objeto de esta clase desde el `main` y llamad a sus métodos.
  En esta solución, asignamos (dentro del constructor) un primer valor a cada uno de los tres datos miembro y posteriormente permitimos su modificación a través del método `SetValores`. Sólo permitimos la modificación conjunta de los tres datos miembro a la misma vez.
- Alternativamente, Creamos la clase
  `Conjunto_3_Valores_vs4`
  que contendrá el mismo código que `Conjunto_3_Valores_vs3`, salvo que suprimimos el constructor de la clase. Modificad convenientemente el `main`.
  En esta solución, los tres datos miembro sólo los podemos asignar con el método nuevo `SetValores`. Sólo permitimos la modificación conjunta de los tres datos miembro a la misma vez.
- Alternativamente, creamos la clase
  `Conjunto_3_Valores_vs5`
  que contendrá el mismo código que `Conjunto_3_Valores_vs4`, pero suprimimos el método `SetValores`, y le añadimos tres métodos nuevos `SetPrimero`, `SetSegundo`, `SetTercero`, para que cambien cada uno de los valores de forma independiente. Cread un objeto de esta clase desde el `main` y llamad a sus métodos.
  En esta solución, los tres datos miembro sólo los podemos asignar a través de un método específico para cada uno de ellos.

Observad la flexibilidad que me ofrece el concepto de clase. El programador decide cómo quiere que se acceda a los datos miembro: prohibiendo cambiar los datos miembro una vez que se ha construido el objeto (versión 2) o permitiéndolo, bien todos a la vez (versiones 3 y 4) o uno a uno (versión 5).

Finalidad: Trabajar con los conceptos básicos de clases. Dificultad Baja.

A partir de ahora, todos los ejercicios deben resolverse utilizando únicamente datos miembro privados.

8. Cread una clase llamada `ParejaEnteros` que represente una pareja de enteros cualesquiera. Añadid un constructor y los métodos que se estimen oportunos para asignarles valores a los enteros y para obtener el valor que éstos tengan. Las operaciones que queremos realizar sobre una pareja son calcular el mayor, el menor, comprobar si se dividen, ver quien es el dividendo y quien el divisor, intercambiarlos y calcular el máximo común divisor. Cread un `main` que llame a dichos métodos, para comprobar que están bien implementados.

   ¿Le añadiría a la clase `ParejaEnteros` un método para calcular el factorial de un número? ¿Por qué?

   Finalidad: Diseñar la interfaz de una clase. Dificultad Baja.

9. Sobre el ejercicio 8 de esta relación de problemas se pide crear un programa principal más completo de la siguiente forma. En el `main` se leerán dos enteros y a continuación se ofrecerá al usuario un menú con las opciones para realizar cualquiera de las operaciones definidas en la clase `ParejaEnteros`. También se ofrecerá la posibilidad de imprimir el valor de cada uno de los dos enteros. Una vez ejecutada la operación, se volverá a mostrar el mismo menú, hasta que el usuario decida salir (pulsando la tecla `'X'`, por ejemplo).

   Finalidad: Organizar un menú de opciones que se va repitiendo mientras el usuario no decida salir. Dificultad Media.

10. Definid una clase para representar una progresión geométrica, tal y como se definió en el ejercicio 19 de la relación de problemas II.

    a) Pensad cual debería ser el constructor de la clase y los métodos *básicos* que la clase debería tener.

    b) Añadid un método llamado `SumaHastaInfinito` para que calcule la suma dada por:

    $$\sum_{i=1}^{i=\infty} a_i = \frac{a_1}{1-r}$$

    siempre que el valor absoluto de la razón sea menor o igual que 1.

    c) Añadid un método llamado `SumaHasta` para que sume los k primeros elementos de la progresión. Para implementar el método `SumaHasta`, usad el mismo algoritmo (con un bucle `for`) que se vio como solución del ejercicio 19 de la relación de problemas II.

    Cread un programa principal que construya la progresión con los valores $a_1 = 3{,}5$ y $r = 4{,}1$ y calcule la suma hasta $k = 30$.

    d) Cambiemos ahora la implementación del método `SumaHasta`. Para ello, en vez de usar un bucle `for` aplicamos la siguiente fórmula que nos da la sumatoria

aplicando únicamente cinco operaciones:

$$\sum_{i=1}^{i=k} a_1 r^i = a_1 \frac{r^k - 1}{r - 1}$$

Es muy importante remarcar que el programa `main` no cambia nada. Hemos cambiado la implementación del método `SumaHasta` y lo hemos podido hacer sin cambiar el `main`, ya que éste no tenía acceso al código que hay dentro del método. Esto es *ocultación de información* tal y como se describió en las clases de teoría.

Nota. La potenciación (en nuestro ejemplo r^k) es una operación costosa, por lo que hasta podría ser más lenta la versión nueva que la antigua usando un bucle `for`. Probad distintos valores para ver si hay diferencias significativas. En cualquier caso, lo importante es que mientras no cambiemos la cabecera del método `SumaHasta`, podemos cambiar su implementación sin tener que cambiar ni una línea de código del `main`.

e) Añadid ahora otro método `MultiplicaHasta` para que multiplique los k primeros elementos de la progresión, aplicando la fórmula:

$$\prod_{i=1}^{i=k} a_1 r^i = \sqrt{(a_1 a_k)^k}$$

Finalidad: Enfatizar la importancia de la ocultación de información. Dificultad Baja.

11. Se quiere construir una clase `DepositoSimulacion` para simular préstamos, ofreciendo la funcionalidad descrita en los ejercicios 11 (reinvierte capital e interés un número de años) y 12 (reinvierte capital e interés hasta obtener el doble de la cantidad inicial) de la relación de problemas II (página 16). Por tanto, la clase debe proporcionar, para un capital y unos intereses dados, métodos para:

    - calcular el capital que se obtendrá al cabo de un número de años,
    - calcular el número de años que deben pasar hasta obtener el doble de la cantidad inicial.

    A la hora de diseñar la clase, tendremos que analizar cuestiones como:

    a) Si queremos modificar el capital y el interés una vez creado el objeto,

    b) Si queremos poder modificarlos de forma independiente,

    c) Si hay alguna restricción a la hora de asignar un valor al capital e interés,

    d) Si es mejor un método para calcular el número de años hasta obtener el doble de la cantidad inicial, o por el contrario es mejor un método para calcular el número de años hasta obtener una cantidad específica.

Finalidad: Diseñar la interfaz de una clase. Dificultad Baja.

12. Se quiere construir una clase para realizar la funcionalidad descrita en el ejercicio 10 de la relación de problemas I sobre la nómina del fabricante y diseñador (página 6). La clase debe proporcionar, al menos, los métodos `SalarioNetoDisenador` y `SalarioNetoFabricante`. El criterio del cálculo del salario es el mismo (el diseñador cobre el doble de cada fabricante) pero además, ahora se le va a aplicar una retención fiscal para el cómputo de los salarios netos. Dicha retención será la misma tanto para el fabricante como para el diseñador.

    Analizad las siguientes posibilidades y elegid la que crea más adecuada, justificándolo:

    a) Creamos un dato miembro para almacenar el valor de retención fiscal y pasamos su valor en el constructor.

    b) Creamos un dato miembro para almacenar su valor y además proporcionamos un método para su modificación.

    Finalidad: Trabajar con datos miembro constantes. Dificultad Baja.

13. Una empresa quiere gestionar las nóminas de sus empleados. El cómputo de la nómina se realiza en base a los siguientes criterios:

    a) Hay cuatro tipos de categorías laborales: Operario, Base, Administrativo y Directivo.

    b) Se parte de un salario base que depende de la antigüedad del trabajador y de su categoría laboral. Para la categoría Operario, el salario base es de 900 euros, 1100 el puesto Base, 1200 los Administrativos y 2000 los Directivos. Dicho salario base se incrementa con un tanto por ciento igual al número de años trabajados.

    c) Los trabajadores tienen complementos en su nómina por el número de horas extraordinarias trabajadas. La hora se paga distinta según la categoría: 16 euros por hora para los operarios, 23 para el puesto Base, 25 los Administrativos y 30 los Directivos. Además, al complemento que sale al computar el número de horas extraordinarias, se le aplica una subida con un tanto por ciento igual al número de años trabajados.

    Se pide diseñar la interfaz de una clase (también hay que incluir los datos miembro privados) para poder trabajar con esta información. No se pide implementar la clase, únicamente determinar la interfaz.

    Finalidad: Diseñar la interfaz de una clase. Dificultad Media.

14. Transformad la solución del ejercicio 9 de esta relación de problemas para que la responsabilidad de la lectura de la opción introducida por el usuario recaiga en una clase a la que llamaremos `Menu`.

    Finalidad: Ver una clase específica que gestione la entrada de datos. Dificultad Media.

15. Cread una clase para representar fracciones. El numerador y denominador serán enteros y debe proporcionar un método para normalizar la fracción, es decir, encontrar la fracción irreducible que sea equivalente. Para ello, tendrá que dividir el numerador y denominador por su máximo común divisor.

    Finalidad: Diseñar la interfaz de una clase. Dificultad Media.

16. En C++, los tipos de datos simples como `int`, `char`, `double`, etc, no son clases. Por tanto, si declaramos una variable real en la forma `double real`, para calcular el seno de `real`, llamamos a una función en la forma

    ```
    double real, resultado;
    real = 6.5;
    resultado = sin(real);
    ```

    Estamos interesados en construir una clase llamada `MiReal` para trabajar con reales al *estilo de las clases*. Por tanto, para calcular el seno de `6.5`, deberíamos hacer algo como lo siguiente:

    ```
    MiReal real(6.5);
    double resultado;
    resultado = real.Seno();
    ```

    En esta caso, `Seno` es un método de la clase `MiReal` que devuelve un tipo `double`. Construid dicha clase, proporcionando los siguientes métodos:

    a) `Valor`. Este método debe devolver el valor del real (6.5 en el ejemplo anterior)

    b) `Seno`. Para definirlo, llamad a la propia función `sin` de la biblioteca `cmath`. Debe devolver un `double`

    c) `ValorAbsoluto`. Devolverá un tipo `double` que contendrá el valor absoluto del real. No pueden utilizarse las funciones de `cmath`.

    Cread un programa principal que llame a dichos métodos.

    Observad que sería interesante que nuestro código fuese el siguiente:

    ```
    MiReal real(6.5);
    MiReal resultado;
    resultado = real.Seno();
    resultado = resultado + real;
    ```

    Es decir, que el método `Seno` devolviese un tipo `MiReal` en vez de un tipo `double`. Aunque esto es posible en C++, por ahora no disponemos de las herramientas necesarias para hacerlo. Por ejemplo, para hacer la asignación entre variables de tipo `MiReal` se necesita el concepto de *sobrecarga del operador de asignación* y para

realizar la suma de dos variables de tipo `MiReal` se necesita *sobrecargar* el operador + (se verá en el segundo cuatrimestre).

Nota. Este tipo de clases (`MiEntero`, `MiReal`, etc) son proporcionadas por Java de forma standard. En otros lenguajes como SmallTalk o plataformas como .NET (con lenguajes como Visual Basic, C#) no existe la diferencia entre tipos *básicos* (como `int`, `double`, etc) y todo son clases, siguiendo una filosofía orientada a objetos *pura*.

Finalidad: Entender las clases que actúan como envolventes (wrappers) de los tipos básicos. Dificultad Baja.

Problemas de Ampliación

17. Definid funciones aisladas para implementar las soluciones a los siguientes ejercicios de la relación de problemas II: 15 (factorial y combinatorio), 21 (serie), 14 (narcisista), 27 (feliz).

    Dificultad Baja.

18. Construid la clase `MiEntero` que será utilizada en la forma siguiente:

    ```
    MiEntero entero(6);
    int entero;
    resultado = entero.Factorial();
    ```

    Debe proporcionar los siguientes métodos:

    a) `Valor`. Este método debe devolver el valor del entero (un tipo `int`).

    b) `Factorial`. Usad la solución del ejercicio 15 de la relación de problemas II (página 17).

    c) `EsNarcisista`. Usad la solución del ejercicio 14 de la relación de problemas II (página 17)

    Cread un programa principal que llame a dichos métodos.

19. Construid la clase `MiCaracter` que será utilizada en la forma siguiente:

    ```
    MiCaracter caracter('a');
    char resultado;
    resultado = entero.GetMayuscula();
    ```

    Debe proporcionar los siguientes métodos:

    a) `Valor`. Este método debe devolver el valor del carácter.

    b) `EsMinuscula`. Devuelve un `bool`.

    c) `EsMayuscula`. Devuelve un `bool`.

    d) `EsLetra`. Devuelve un `bool`.

    e) `GetMayuscula`. Devuelve un `char` correspondiente al carácter convertido a mayúscula. Si el carácter no es minúscula, devuelve el mismo valor del `char`.

    f) `GetMinuscula`. El recíproco del anterior.

    Cread un programa principal que llame a dichos métodos.

20. La sonda Mars Climate Orbiter fue lanzada por la NASA en 1998 y llegó a Marte el 23 de septiembre de 1999. Lamentablemente se estrelló contra el planeta ya que se acercó demasiado. El error principal fue que los equipos que desarrollaron los distintos módulos de la sonda usaron sistemas de medida distintos (el anglosajón y el métrico). Cuando un componente software mandaba unos datos en millas (o libras), otro componente software los interpretaba como si fuesen kilómetros (o Newtons). El problema se habría arreglado si todos hubiesen acordado usar el mismo sistema. En cualquier caso, cada equipo se encuentra más a gusto trabajando en su propio sistema de medida. Por tanto, la solución podría haber pasado por que todos utilizasen una misma clase para representar distancias (idem para fuerzas, presión, etc), utilizando los métodos que les resultasen más cómodos.

    Para ello, se pide construir la clase `Distancia` que contendrá métodos como `SetKilometros`, `SetMillas`, etc. Internamente se usará un único dato miembro privado llamado `kilometros` al que se le asignará un valor a través de los métodos anteriores, realizando la conversión oportuna (una milla es $1{,}609344$ kilómetros). La clase también proporcionará métodos como `GetKilometros` y `GetMillas` para lo que tendrá que realizar la conversión oportuna (un kilómetro es $0{,}621371192$ millas).

    Observad que la implementación de la clase podría haber utilizado como dato miembro privado, una variable `millas`, en vez de `kilómetros`. Esto se oculta a los usuarios de la clase, que sólo ven los métodos `SetKilometros`, `SetMillas`, `GetKilometros` y `GetMillas`.

    Nota. Otro de los fallos del proyecto fue que no se hicieron suficientes pruebas del software antes de su puesta en marcha, lo que podría haber detectado el error. Esto pone de manifiesto la importancia de realizar una batería de pruebas para comprobar el correcto funcionamiento del software en todas las situaciones posibles. Esta parte en el desarrollo de un proyecto se le conoce como *pruebas de unidad (unit testing)*

    Finalidad: Trabajar con una clase como una abstracción de un concepto. *Dificultad Baja.*

21. Se quiere construir una clase para representar la tracción de una bicicleta, es decir, el conjunto de estrella, cadena y piñón. Supondremos que el plato delantero tiene tres estrellas y el trasero tiene 7 piñones. La clase debe proporcionar métodos para cambiar la estrella y el piñón, sabiendo que la estrella avanza o retrocede de 1 en 1, y los piñones cambian a salto de uno o de dos.

    Una vez hecha la clase y probada con un programa principal, modificadla para que no permita cambiar la marcha (con la estrella o el piñón) cuando haya riesgo de que se rompa la cadena. Este riesgo se produce cuando la marcha a la que queremos cambiar es de la siguiente forma:

    - Primera estrella y piñón mayor o igual que 5
    - Segunda estrella y piñón o bien igual a 1 o bien igual a 7
    - Tercera estrella y piñón menor o igual que 4

Finalidad: Diseñar la interfaz de una clase. Trabajar con datos miembro constantes.. Dificultad Media.

22. Construid una clase llamada `MedidaAngulo` que represente una medida de un ángulo. Al igual que se hizo en el ejercicio 20, la clase aceptará datos que vengan de alguna de las siguientes formas: número de grados con decimales (real); número de radianes (entero); número de segundos (entero); número de grados, minutos y segundos (en un struct que represente estos tres valores)

    Dificultad Baja.

23. Recuperad el ejercicio 5 de esta relación de problemas sobre la función gaussiana. Ahora estamos interesados en obtener el área que cubre la función en el intervalo $[-\infty, x]$. Dicho valor se conoce como la *distribución acumulada (cumulative distribution function)* en el punto x, abreviado $CDF(x)$. Matemáticamente se calcula realizando la integral $\int_{-\infty}^{x} \text{gaussiana}(x)dx$, tal y como puede comprobarse ejecutando el applet disponible en:
    `http://dostat.stat.sc.edu/prototype/calculators/index.php3?dist=Normal`

    También pueden obtenerse un valor aproximado de $CDF(x)$, como por ejemplo, el dado por la siguiente fórmula (ver Wikipedia: Normal distribution)

    $$CDF(x) = \text{Área hasta } (x) \approx 1 - \text{gaussiana}(x)(b_1 t + b_2 t^2 + b_3 t^3 + b_4 t^4)$$

    dónde:

    $$t = \frac{1}{1 + b_0 x} \quad b_0 = 0{,}2316419 \;\; b_1 = 0{,}319381530 \;\; b_2 = -0{,}356563782$$

    $$b_3 = 1{,}781477937 \;\; b_4 = -1{,}821255978 \;\; b_5 = 1{,}330274429$$

    Para implementarlo, podríamos definir la función siguiente:

    ```
    double AreaHasta (double esperanza, double desviacion, double x)
    ```

    En vez de trabajar con funciones, planteamos la solución con una clase. La clase se llamará `Gaussiana` y crearemos objetos pasándole en el constructor la esperanza y la distribución (los parámetros que definen la gaussiana). Le añadiremos métodos para:

    - Obtener el valor de la función en un punto x cualquiera.
    - Obtener el área hasta un punto x cualquiera.

    Dificultad Media.

24. Cread un `struct` llamado `CoordenadasPunto2D` para representar un par de valores reales correspondientes a un punto en $\mathbb{R}^2$. Cread ahora una clase llamada `Circunferencia`. Para establecer el centro, se usará un valor del tipo `CoordenadasPunto2D`. Añadid métodos para obtener la longitud de la circunferencia y el área del círculo interior. Añadid también un método para saber si la circunferencia contiene a un punto. Recordemos que un punto (x_1, y_1) está dentro de una circunferencia con centro (x_0, y_0) y radio r si se verifica que:

    $$(x_0 - x_1)^2 + (y_0 - y_1)^2 <= r^2$$

    Observad que el valor de π debe ser constante, y el mismo para todos los objetos de la clase `Circunferencia`.

    Finalidad: Trabajar con constantes estáticas de tipo `double`. *Dificultad Baja.*

25. Implementad la clase del ejercicio 13 de esta relación de problemas. *Dificultad Media.*

Sesión 6

► Actividades a realizar en casa

Funciones y Clases

Actividad: Resolución de problemas.

Realizad una lectura rápida de las actividades a realizar durante la próxima sesión de prácticas en las aulas de ordenadores.

Resolved los ejercicios siguientes de la relación de problemas III de Funciones y Clases.

- *Obligatorios:*

1 `solucion1.cpp`

2 `solucion2.cpp`

3 `solucion3.cpp`

5 `solucion5.cpp`

- *Opcionales:*

17II **Relacion II** Calcular mediante un programa en C++ el combinatorio de n, m **pero utilizando funciones**, tampoco se puede usar las funciones de la biblioteca cmath. `solucion17II.cpp`

6 `solucion6.cpp`