



Fundamentos de Programación.

Guión de Prácticas.

Curso 2016/2017

Para cualquier sugerencia o comentario sobre este guión de prácticas, por favor, enviad un e-mail a Juan Carlos Cubero (JC.Cubero@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 guón de prácticas

El guón está dividido en sesiones. En cada sesión se plantean una serie de problemas de programación a resolver. En la semana número *i* se publicará la **Sesión *i***. En dicha sesión se especifica la lista de problemas que el alumno tiene que resolver.

Las soluciones de los ejercicios deberán ser subidas a la plataforma de decsai, en el plazo que el profesor determine. Para ello, el alumno debe entrar en el acceso identificado de decsai, seleccionar **Entrega Prácticas** y a continuación la práctica correspondiente a la semana en curso. El alumno subirá un fichero **zip** que contendrá los ficheros con extensión **cpp** correspondientes a las soluciones de los ejercicios.

La defensa de la sesión *i* se hará la semana siguiente (semana *i + 1*), durante las horas de prácticas. El profesor llamará aleatoriamente a los alumnos para que defiendan dichos ejercicios (a veces explicándolos a sus compañeros) Simultáneamente a la defensa, todos los alumnos tendrán que ir realizando una serie de actividades que vienen descritas en este guón. Dichas actividades no se entregarán al profesor. Terminada la defensa, el profesor explicará los ejercicios a todos los alumnos. Además, al final de la semana, las soluciones estarán disponibles en decsai. Es muy importante que el alumno revise estas soluciones y las compare con las que él había diseñado.

Los problemas a resolver en cada sesión están incluidos en las *Relaciones de Problemas*. Hay una relación de problemas por cada tema de la asignatura. Los problemas que hay que entregar son de dos tipos:

1. **Obligatorios**: Todos los alumnos deben resolver estos problemas.

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

2. **Opcionales**: Su entrega no es obligatoria.

Si se realizan correctamente estos ejercicios, el alumno podrá sacar hasta un 10 (sobre 10) en la nota de prácticas. Para poder optar a la Matrícula de Honor es necesario realizar todos los ejerciciosopcionales.

Para la realización de estas prácticas, se utilizará el entorno de programación Orwell Dev C++. En la página 3 se encuentran las instrucciones para su instalación en nuestra casa. En cualquier caso, el alumno puede instalar en su casa cualquier otro compilador.

Muy importante:

- La resolución de los problemas y actividades puede hacerse en grupo, pero la defensa durante las horas de prácticas es individual.
- Es muy importante que la asignatura se lleve al día para poder realizar los ejercicios propuestos en estos guones.

Instalación de Orwell Dev C++ en nuestra casa

El entorno de desarrollo que usaremos será Orwell Dev C++. Puede descargarse desde la página:

http://sourceforge.net/projects/orwelldevcpp/?source=typ_redirect

Cuando lo instalemos en nuestra casa, configurar las siguientes opciones:

Herramientas -> Opciones del Compilador

Compilador a configurar: TDM-GCC ... Debug

Configuración -> Code Warnings. Marcar los siguientes:

Show most warnings

Show some more warnings

Configuración -> Linker.

Generar información de Debug: Yes

Generación de código -> Language standard (-std) -> ISO C++11

Herramientas -> Opciones del editor

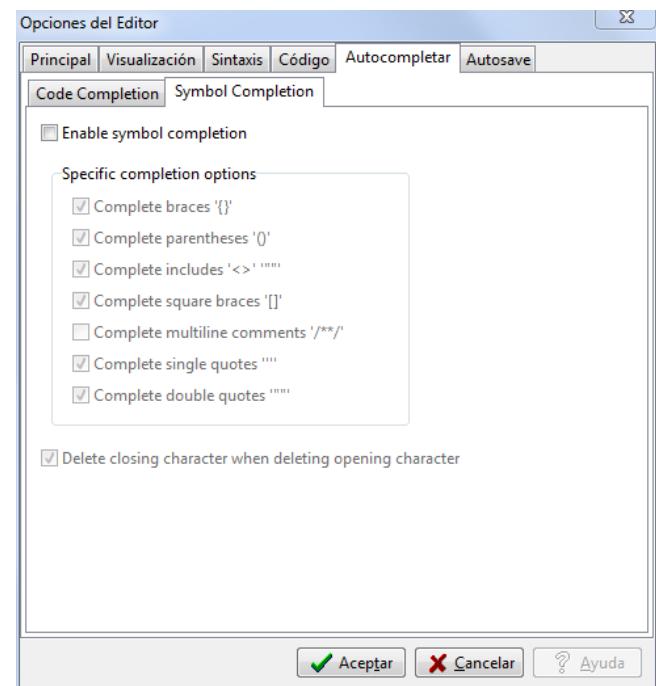
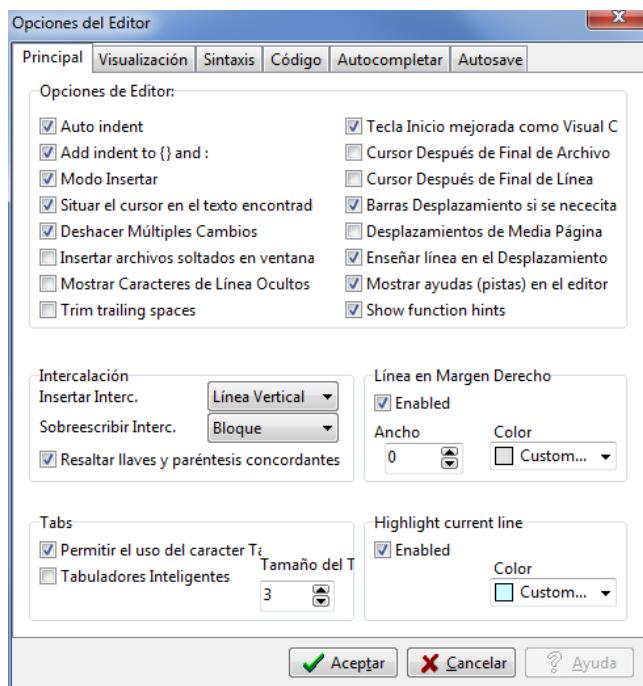
-> Principal

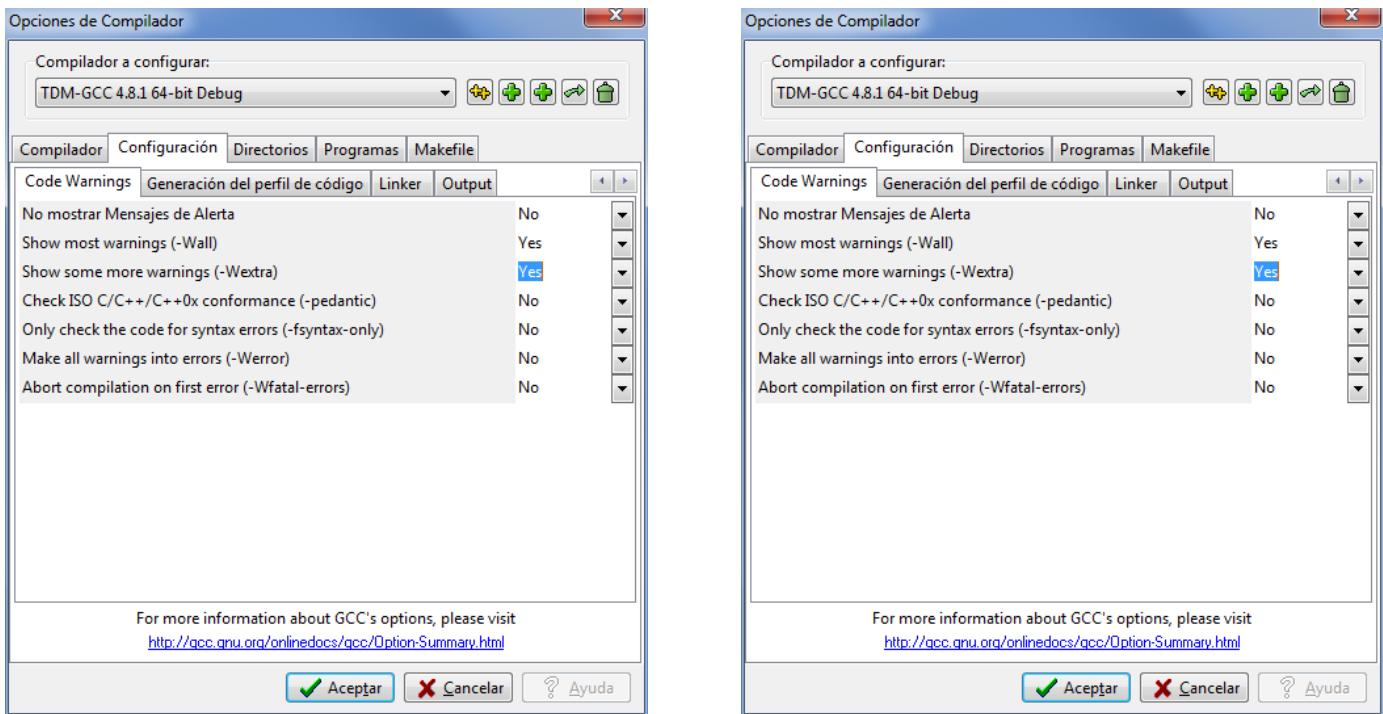
Desmarcar Tabuladores inteligentes

Tamaño del tabulador: 3

-> Autocompletar -> Symbol completion

Desmarcar Enable Symbol completion





Preparar y acceder a la consola del sistema

La consola de Windows (la ventana con fondo negro que aparece al ejecutar el comando `cmd.exe`, o bien la que sale al ejecutar un programa en Dev C++) no está preparada por defecto para mostrar adecuadamente caracteres latinos como los acentos. Por ejemplo, al ejecutar la sentencia de C++

```
cout << "Atención"
```

saldrá en la consola un mensaje en la forma

`Atenci3/4n`

Para que podamos ver correctamente dichos caracteres, debemos seguir los siguientes pasos:

1. Cambiar la fuente de la consola a una que acepte caracteres Unicode. En la versión de XP de las aulas ya se ha realizado dicho cambio. En nuestra casa, tendremos que hacer lo siguiente:

`Inicio -> Ejecutar -> cmd`

Una vez que se muestre la consola, hacemos click con la derecha y seleccionamos **Predeterminados**. Seleccionamos la fuente **Lucida Console** y aceptamos.

2. Debemos cargar la página de códigos correspondiente al alfabeto latino. Para ello, tenemos varias alternativas:

- a) Incluir la siguiente sentencia al inicio de nuestro programa (.cpp), al empezar el `main`:

```
setlocale(LC_ALL, "spanish");
```

- b) Lo siguiente es sólo para el SO Microsoft Windows. Si queremos que la consola siempre cargue la tabla de caracteres latinos, debemos modificar el registro de Windows. Lo abrimos desde

Inicio->Ejecutar->regedit

Nos situamos en la clave

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\  
    \Control\Nls\CodePage
```

y cambiamos el valor que hubiese dentro de OEMCP y ACP por el de 1252. Esta es una página muy parecida a la 8859-1 referenciada en los apuntes (Tema I)

Esta es la forma recomendada y la que se ha usado en las aulas de prácticas. Requiere reiniciar el ordenador.

Muy Importante: Si se usa otra tabla (distinta a 1252), el sistema operativo podría incluso no arrancar.

Tabla resumen de accesos directos usados en Orwell Dev C++

Tab	Tabula una línea o un bloque
Shift Tab	Quita tabulación a una línea o un bloque
Ctrl Barra Espaciadora	Ayuda autocompletación del código
F9	Compilar
F10	Ejecutar
F11	Compilar y Ejecutar
F5	Depurar
	Empieza la depuración
F7	Siguiente paso
	Ejecución paso a paso sin entrar en los métodos o funciones
F8	Avanzar paso a paso
	Ejecución paso a paso entrando en los métodos o funciones

Sesión 1

Tal y como se ha indicado al inicio de este documento, en la primera semana de clase se publica la sesión 1. En esta sesión se detalla las tarea y ejercicios que el alumno debe resolver en su casa durante la primera semana y que defenderá en la siguiente. Esta es la única sesión en la que el alumno no tendrá que entregar las soluciones a través de decsai.

► **Actividades a realizar en casa**

Actividad: Conseguir login y password.

El alumno debe registrarse electrónicamente como alumno de la Universidad, tal y como se indica en el fichero de información general de la asignatura. De esta forma, obtendremos un login y un password que habrá que introducir al arrancar los ordenadores en las aulas de prácticas. La cuenta tarda 48 horas an activarse, por lo que el registro debe realizarse al menos dos días antes de la primera sesión de prácticas.

Actividad: Instalación de Orwell Dev C++.

Durante la primera semana de clase, el alumno debería instalar en su casa el compilador Orwell Dev C++. Consulte la sección de Instalación (página 3) de este guión.

Actividad: Resolución de problemas.

Resuelva en papel los ejercicios siguientes de la relación de problemas I:

- *Obligatorios:*

- 1 (Asignaciones secuenciales)
- 3 (Circunferencia)
- 4 (Subir sueldo)
- 5 (Interés bancario)

- *Opcionales:*

- 7 (Triles: intercambiar variables)

Actividad: Preparar la clase de prácticas de la semana próxima.

Realice una lectura rápida de las actividades a realizar la semana próxima durante las horas de prácticas en las aulas de ordenadores (ver páginas siguientes)



Actividades de Ampliación

Lea el artículo de Norvig: *Aprende a programar en diez años*

<http://lоро.sourceforge.net/notes/21-dias.html>

sobre la dificultad del aprendizaje de una disciplina como la Programación.

► **Actividades a realizar en las aulas de ordenadores**

Estas son las actividades que se realizarán durante las clases de prácticas en la segunda semana de clase.

El Entorno de Programación. Compilación de Programas

Arranque del Sistema Operativo

Para poder arrancar el SO en las aulas de ordenadores, es necesario obtener el login y password indicados en las actividades a realizar en casa.

En la casilla etiquetada como Código, introduciremos fp. Al arrancar el SO, aparecerá una instalación básica de Windows con el compilador Orwell Dev C++. Todo lo que escribamos en la unidad C: se perderá al apagar el ordenador.

Por ello, el alumno dispone de un directorio de trabajo en la unidad lógica U:, cuyos contenidos permanecerán durante todo el curso académico. En cualquier caso, es recomendable no saturar el espacio usado ya que, en caso contrario, el compilador podría no funcionar.

El alumno deberá crear el directorio U:\FP dentro de su unidad U:

Para acceder a la unidad U: desde nuestras casas, debemos usar cualquier programa de ftp que use el protocolo ssh, como por ejemplo filezilla o winscp. Instalamos este programa en nuestra casa y simplemente nos conectamos a turing.ugr.es con nuestras credenciales.

El primer programa

Copiando el código fuente

Descargue de decsai el fichero I_Pitagoras y cópielo en su carpeta local (dentro de U:\FP). También puede bajarse el fichero desde la siguiente dirección:

http://decsai.ugr.es/~carlos/FP/I_Pitagoras.cpp

Desde el Explorador de Windows, haga doble click sobre el fichero I_Pitagoras.cpp. Debe aparecer una ventana como la de la figura 1

The screenshot shows a Dev-C++ IDE window with the following details:

- Title Bar:** K:\doc_GoogleDrive_\FUNDAMENTOS_DE_PROGRAMACION_Material Web Inicial\fp\Proyecto Pitágoras_\Pitagoras_\Pitagoras.cpp - Dev-C++ 5.6.3
- Menu Bar:** Archivo, Edición, Buscar, Ver, Proyecto, Ejecutar, Herramientas, CVS, Ventana, Ayuda.
- Toolbar:** Includes icons for file operations like Open, Save, Print, and Build, along with search and filter tools.
- Build Line:** TDM-GCC 4.8.1 64-bit Debug
- Project View:** Shows a single file I_Pitagoras.cpp selected.
- Code Editor:** Displays the following C++ code:

```
/* Programa para calcular la hipotenusa de un triángulo.
   Implementa el algoritmo de Pitágoras
   Necesita: Los catetos de un triángulo rectángulo
   lado1, lado2.
   Calcula: La hipotenusa, calculada como
   hip = RaizCuad(lado1^2 + lado2^2)
*/
#include <iostream> // Inclusión de los recursos de E/S
#include <cmath> // Inclusión de los recursos matemáticos
using namespace std;
int main(){
    double lado1; // Declara variables para guardar
    double lado2; // los dos lados y la hipotenusa
    double hip;
    cout << "Introduzca la longitud del primer cateto: ";
    cin >> lado1;
    cout << "Introduzca la longitud del segundo cateto: ";
    cin >> lado2;
    hip = sqrt(lado1*lado1 + lado2*lado2);
    cout << "\nLa hipotenusa vale " << hip << "\n\n";
}
```

- Status Bar:** Line: 1 Col: 28 Sel: 0 Lines: 27 Length: 899 Insertar Modificado

Figura 1: Programa que implementa el algoritmo de Pitágoras

Algunas consideraciones con respecto a la escritura de código en C++ (ver figura 2)

- Es bueno que, desde el principio se incluyan comentarios indicando el objetivo del programa y resaltando los aspectos más importantes de la implementación.
- Es muy importante una correcta tabulación de los programas. Por ahora, incluiremos todas las sentencias del programa principal con una tabulación. Sólo es necesario incluir la primera; el resto las pone automáticamente el entorno, al pasar a la siguiente línea.
- Para facilitar la lectura del código fuente, se deben usar espacios en blanco para separar las variables en la línea en la que van declaradas, así como antes y después del símbolo = en una sentencia de asignación. Dejad también un espacio en blanco antes y después de << y >> en las sentencias que contienen una llamada a cout y cin respectivamente.

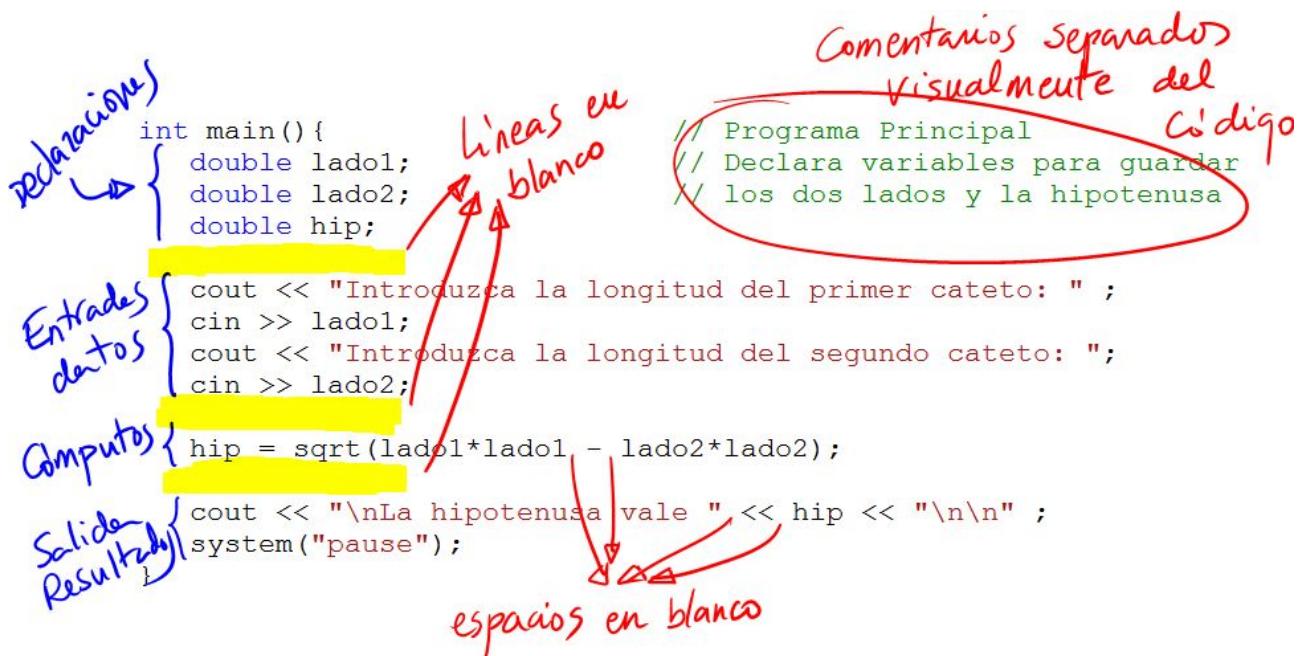


Figura 2: Escritura de código

No respetar las normas de escritura de código baja puntos en todos los exámenes y prácticas de la asignatura

IMPORTANT

Compilación

Una vez cargado el programa, pasamos a comprobar si las sentencias escritas son sintácticamente correctas, es decir, pasamos a *compilar* el programa. Para ello pulsamos F9, o bien sobre el icono .

Para que el proceso de compilación se realice de forma correcta y se obtenga el programa ejecutable, es necesario que el código fuente no contenga errores sintácticos. Si aparecen errores, es necesario volver a la fase de edición, guardar de nuevo el código fuente y repetir la fase de compilación.

Como resultado de la fase de compilación, en la parte de abajo del entorno debe aparecer un mensaje del tipo:

Compilation succeeded

Una vez compilado el programa, habremos obtenido el fichero **I_Pitagoras.exe**. Para ejecutarlo desde el entorno basta pulsar sobre F10. Si se quiere, ambos pasos (compilación y ejecución) pueden realizarse pulsando sobre F11. Debe aparecer una ventana de comandos del Sistema, en la que se estará ejecutando el programa. La ejecución del programa se detendrá en aquellos puntos del mismo donde se requiera la interacción del usuario para poder proseguir, es decir, en las operaciones de entrada de datos a través del dispositivo estándar de entrada. En este ejemplo, sería en las dos operaciones `cin`. En el resto de los casos, la ejecución del programa continuará hasta el final. La introducción de datos mediante la sentencia `cin` se hace siempre de la misma manera; primero se introduce el valor que se desee y al terminar se pulsa la tecla RETURN.

Introducid ahora los valores pedidos en el ejemplo de Pitágoras y comprobad la respuesta del programa.

Como hemos indicado anteriormente, en la fase de generación del ejecutable se ha creado un fichero en el Sistema que se llama igual que nuestro fichero pero sustituyendo la extensión "cpp" por "exe", es decir, **I_Pitagoras.exe**. Este fichero se encuentra en el mismo directorio que el del fichero `cpp`. Para mostrar que el fichero generado es independiente del entorno de programación, hacemos lo siguiente:

1. Cerramos Orwell Dev C++.
2. Abrid una ventana de Mi PC.
3. Situarse en la carpeta que contiene el ejecutable.
4. Haced doble click sobre el fichero **I_Pitagoras.exe**.

Prueba del programa

Uno podría pensar que una vez que consigo un fichero ejecutable a partir de mi código fuente, el problema está terminado. Sin embargo esto no es así. Tras el proceso de compilado se

requiere una fase de prueba. Dicha fase intenta probar que el algoritmo planteado resuelve el problema propuesto. Para llevar a cabo esta fase, es necesario ejecutar el programa y verificar que los resultados que obtiene son los esperados.

Ahora que podemos ver el resultado obtenido por el programa implementado, verifiquemos mediante el siguiente conjunto de pruebas que el programa funciona de forma correcta.

lado1	lado2	hip
3	4	5
1	5	5.099
2.7	4.3	5.077
1.25	2.75	3.02

Una vez que el algoritmo supera la fase de prueba, podemos considerar que se ha concluido con la fase inicial del desarrollo del software.

Introducción a la corrección de errores

Los errores de compilación

Ya hemos visto los pasos necesarios para construir un fichero ejecutable a partir del código fuente. El paso central de este proceso era la fase de compilación. En esta parte de este guión de prácticas aprenderemos a corregir los errores más comunes que impiden una compilación exitosa del fichero fuente.

Cargad el fichero **I_Pitagoras.cpp**. Quitadle una '**u**' a alguna aparición de **cout**. Intentad compilar. Podemos observar que la compilación no se ha realizado con éxito. Cuando esto sucede, en la parte inferior de la ventana principal aparecen los errores que se han encontrado. Aparece una descripción del error, así como otra información, como el número de línea en la que se produjo. Los pasos que debemos seguir para la corrección son los siguientes:

1. Ir a la primera fila de la lista de errores.
2. **Leer el mensaje de error e intentar entenderlo.**
3. Hacer doble click sobre esa fila con el ratón. Esto nos posiciona sobre la línea en el fichero fuente donde el compilador detectó el error.
4. Comprobar la sintaxis de la sentencia que aparece en esa línea. Si se detecta el error, corregirlo. Si no se detecta el error mirar en la línea anterior, comprobar la sintaxis y repetir el proceso hasta encontrar el error.
5. Después de corregir el posible error, guardamos de nuevo el archivo y volvemos a compilar. Esto lo hacemos aunque aparezcan más errores en la ventana. La razón es que es posible que el resto de los errores sean consecuencia del primer error.

6. Si después de corregir el error aparecen nuevos errores, volver a repetir el proceso desde el paso 1.

A veces, el compilador no indica la línea exacta en la que se produce el error, sino alguna posterior. Para comprobarlo, haced lo siguiente:

- Comentad la línea de cabecera `#include <iostream>` desde el principio. El compilador no reconocerá las apariciones de `cin` o `cout`.
- Quitar un punto y coma al final de alguna sentencia. Dará el error en la línea siguiente.

Para familiarizarnos con los errores más frecuentes y su corrección vamos a realizar el siguiente proceso: a partir del código fuente del ejemplo `I_Pitagoras.cpp`, iremos introduciendo deliberadamente errores para conocer los mensajes que nos aparecen. A continuación se muestran algunos errores posibles. No deben introducirse todos ellos a la vez, sino que han de probarse por separado.

1. Cambiad algún punto y coma por cualquier otro símbolo
2. Cambiad `double` por `dpublic`
3. Cambiad la línea `using namespace std;` por `using namespace STD;`
4. Poned en lugar de `iostream`, el nombre `iotream`.
5. Borrard alguno de los paréntesis de la declaración de la función `main`
6. Introducid algún identificador incorrecto, como por ejemplo `cour`
7. Usad una variable no declarada. Por ejemplo, en la definición de variables cambiad el nombre a la variable `lado1` por el identificador `lado11`.
8. Borrard alguna de las dobles comillas en una constante de cadena de caracteres, tanto las comillas iniciales como las finales.
9. Borrard alguna de las llaves que delimitan el inicio y final del programa.
10. Borrard la línea `using namespace std;` (basta con comentarla con `//`)
11. Cambiad un comentario iniciado con `//`, cambiando las barras anteriores por las siguientes `\\"`
12. Cambiad la aparición de `<<` en `cout` por las flechas cambiadas, es decir, `>>`. Haced lo mismo con `cin`.
13. Suprimid todo el `main`. No hace falta borrar el código, basta con comentarlo.

Además de los errores, el compilador puede generar *aviso*s. Estos se muestran como **Warning** en la misma ventana de la lista de errores. Estas advertencias indican que algún código puede generar problemas durante la ejecución. Por ejemplo, al usar una variable que todavía no tiene un valor asignado, al intentar asignar un entero *grande* a un entero *chico*, etc. Sin embargo, no son errores de compilación, por lo que es posible generar el programa ejecutable correspondiente.

Los errores lógicos y en tiempo de ejecución

Aunque el programa compile, esto no significa que sea correcto. Puede producirse una excepción durante la ejecución, de forma que el programa terminará bruscamente (típico error en Windows de *Exception Violation Address*) o, lo que es peor, dará una salida que no es correcta (error lógico).

Sobre el programa `I_Pitagoras.cpp`, haced lo siguiente:

- Cambiad la sentencia

```
sqrt(lado1*lado1 + lado2*lado2) por:  
sqrt(lado1*lado2 + lado2*lado1)
```

Ejecutad introduciendo los lados 2 y 3. El resultado no es correcto, pero no se produce ningún error de compilación ni en ejecución. Es un error lógico.

- Para mostrar un error de ejecución, declarad tres variables **ENTERAS** (**tipo int**) **resultado**, **numerador** y **denominador**. Asignadle cero a **denominador** y 7 a **numerador**. Asignadle a **resultado** la división de **numerador** entre **denominador**. Imprimid el resultado. Al ejecutar el programa, se produce una excepción o error de ejecución al intentar dividir un entero entre cero.

Creación de un programa nuevo

En esta sección vamos a empezar a crear nuestros propios programas desde Orwell Dev C++. El primer ejemplo que vamos a implementar corresponde al ejercicio 2 sobre la Ley de Ohm, de la relación de problemas I.

Para crear un programa nuevo, abrimos Orwell Dev C++y elegimos

Archivo->Nuevo Código Fuente (Ctr-N)

Para cambiar el nombre asignado por defecto, seleccionamos Archivo -> Guardar Como. Nos vamos a la carpeta U:\FP e introducimos el nombre `I_Voltaje`.

Confirmad que en la esquina superior derecha está seleccionada la opción de compilación

TDM-GCC ... Debug



Figura 3: Creación de un programa nuevo

Ya estamos en condiciones de resolver el problema pedido. Escribimos el código en la ventana de edición. Habrá que leer desde teclado los valores de intensidad y resistencia y el programa imprimirá en pantalla el voltaje correspondiente. Recordad que compilamos con F9 y ejecutamos con F10, o directamente ambas acciones con F11.

Nota. Cuando tenemos varias variables en el código, podemos empezar a escribir el nombre de alguna de ellas y antes de terminar, pulsar Ctr-Barra espaciadora. La ayuda nos mostrará los identificadores disponibles que empiecen por las letras tecleadas.

Resuelva el ejercicio 6 (Diferencia entre instantes de tiempo)

Este ejercicio no tiene que entregarlo, pero lo puede guardar en su unidad en `decsai`. La solución también estará disponible en `decsai` al final de la semana.

Sesión 2

Tipos básicos y operadores

► Actividades a realizar en casa

Actividad: Lectura de programas resueltos.

Lea la solución de los siguientes ejercicios (disponible en decsai)

8 (Conversión sistema métrico)

Actividad: Resolución de problemas.

Resuelva los siguientes problemas de la relación I. Recuerde que antes del domingo por la noche hay que subir las soluciones a decsai, tal y como se explica en la página 2. Debe subir un fichero llamado **sesion2.zip** que incluya todos los ficheros **cpp** (pero **no** los **.exe**).

- *Obligatorios:*

9 (Dos subidas de sueldo)

10 (Gaussiana)

11 (Uso de constantes)

12 (Población)

- *Opcionales:*

13 (Pinta dígitos)

Actividades de Ampliación



Recuerde los conceptos de combinación y permutación, que irán apareciendo recurrentemente a lo largo de la carrera. Consulte, por ejemplo, la siguiente web, para una introducción básica a los conceptos:

<http://www.disfrutalasmaticas.com/combinatoria/combinaciones-permutaciones.html>

Si por ejemplo queremos ver las posibles combinaciones (con repetición e importando el orden) de dos elementos (0 y 1) en 4 posiciones de memoria (4 bits) obtenemos un total de

$2^4 = 16$ posibilidades: 0000, 0001, 0010, …, 1111. Ejecute el siguiente applet para ver las combinaciones resultantes:

<http://dm.udc.es/elearning/Applets/Combinatoria/index.html>

► ***Actividades a realizar en las aulas de ordenadores***

El profesor irá corrigiendo individualmente (a algunos alumnos elegidos aleatoriamente) los ejercicios indicados en la página anterior. Mientras tanto, el resto de alumnos deben intentar resolver los ejercicios de la próxima sesión de prácticas.

Sesión 3

► **Actividades a realizar en casa**

Actividad: Lectura de programas resueltos.

Lea la solución de los siguientes ejercicios (disponible en decsai)

- 20 (Índice de mayúscula)
- 21 (Pasar de carácter a entero)
- 22 (Expresiones diversas)

Actividad: Resolución de problemas.

Resuelva los siguientes problemas de la relación de Problemas I:

- *Obligatorios:*

- 23 (Horas, minutos, segundos)
- 24 (Intercambiar tres variables)
- 25 (Pasar de mayúscula a minúscula)
- 27 (Expresiones lógicas)

- *Opcionales:*

- 28 (Precisión y desbordamiento)

Actividades de Ampliación

Hojar la página

<http://catless.ncl.ac.uk/Risks>

que publica periódicamente casos reales en los que un mal desarrollo del software ha tenido implicaciones importantes en la sociedad.



► **Actividades a realizar en las aulas de ordenadores**

Redireccionando las entradas de cin

Cada vez que se ejecuta `cin`, se lee un dato desde el periférico por defecto. Nosotros lo hemos hecho desde el teclado, introduciendo un dato y a continuación un ENTER. Otra forma alternativa es introducir un dato y luego un separador (uno o varios espacios en blanco o un tabulador). Esto es posible gracias a la existencia de un buffer intermedio que canaliza el flujo de datos entre el programa y la consola. Para más detalle, consulte el final del Tema 1, disponible en [decsai](#).

Para comprobarlo, copiad localmente el fichero `II_cin` disponible en [decsai](#). Observad el código del programa y ejecutadlo. Para introducir los datos pedidos (un entero y dos caracteres) siempre hemos introducido cada valor y a continuación ENTER. Ahora lo hacemos de otra forma alternativa: introducimos los datos separados por espacios en blanco y pulsamos ENTER al final (una sola vez).

Para comprobar el correcto funcionamiento de nuestros programas, tendremos que ejecutarlos en repetidas ocasiones usando distintos valores de entrada. Este proceso lo repetiremos hasta que no detectemos fallos. Para no tener que introducir los valores pedidos uno a uno, podemos recurrir a un simple copy-paste. Para comprobarlo, cread un fichero de texto con un entero y dos caracteres. Separad estos tres datos con varios espacios en blanco. Seleccionad con el ratón los tres y copiadlos al portapapeles (Click derecho-Copiar). Ejecutad el programa y cuando aparezca la consola del sistema haced click derecho sobre la ventana y seleccionad Editar-Pegar.

Otra alternativa es ejecutar el fichero `.exe` desde el sistema operativo y redirigir la entrada de datos al fichero que contiene los datos. Para poder leer los datos del fichero, basta con ejecutar el programa `.exe` desde una consola del sistema y especificar que la entrada de datos será desde un fichero a través del símbolo de redireccionamiento `<` (no ha de confundirse con el token `<<` que aparecía en una instrucción `cout` de C++) Hay que destacar que este redireccionamiento de la entrada lo estamos haciendo en la llamada al ejecutable desde la consola del sistema operativo¹. Para probarlo, descargad desde [decsai](#) el fichero `II_cin_datos_entrada.txt` y copiadlo dentro de la misma carpeta en la que se ha descargado el programa `II_cin`. Abrimos dicha carpeta desde el explorador y seleccionamos con el click derecho del ratón "Abrir Símbolo del Sistema"². Introducimos la instrucción siguiente:

¹También pueden leerse datos de un fichero desde dentro del propio código fuente del programa, pero esto se verá en el segundo cuatrimestre

²Para poder lanzar una consola desde el explorador de Windows, en nuestra casa, o bien instalamos un programa que permita abrir una consola en el directorio actual, como por ejemplo *Open Command Prompt Shell Extension* disponible en <http://code.kliu.org/cmdopen/> o bien abrimos un símbolo del sistema (**Inicio->Ejecutar->cmd**) y vamos cambiando de directorio con la orden `cd`

```
II_cin.exe < II_cin_datos_entrada.txt
```

Ahora, cada vez que se ejecute una instrucción `cin` en el programa, se leerá un valor de los presentes en el fichero de texto.

Cuando ejecutemos el programa, cada ejecución de `cin` leerá un dato desde el fichero indicado, saltándose todos los espacios en blanco y tabuladores que hubiese previamente. Cuando llegue al final del fichero, cualquier entrada de datos posterior que realicemos dará un *fallo*.

Resuelva los siguientes ejercicios de la próxima sesión de prácticas: **5** (Subir sueldo -una única subida salarial-) y **6** (Subir sueldo -dos subidas salariales compatibles-)

Sesión 4

Estructura condicional

► **Actividades a realizar en casa**

Actividad: Lectura de programas resueltos.

Lea la solución de los siguientes ejercicios (disponible en decsai)

- 1 (Media aritmética)
- 2 (Pasar de mayúscula a minúscula)
- 3 (Se dividen)
- 4 (Pasar de mayúscula a minúscula y viceversa)

Actividad: Resolución de problemas.

Resuelva los siguientes problemas de la relación II.

Importante: En estos ejercicios se permite mezclar E/S con cómputos dentro del mismo condicional

- *Obligatorios:*
 - 5 (Subir sueldo -una única subida salarial-)
 - 6 (Subir sueldo -dos subidas salariales compatibles-)
 - 7 (Tres valores ordenados)
 - 8 (Año bisiesto)
- *Opcionales:*
 - 10 (Párking)

► **Actividades a realizar en las aulas de ordenadores**

En esta sesión empezaremos a trabajar en el aula con las estructuras condicionales. Es muy importante poner atención a la tabulación correcta de las sentencias, tal y como se indica en las transparencias. Recordad que una tabulación incorrecta supondrá bajar puntos en la primera prueba práctica que se realizará dentro de algunas semanas.

El entorno de compilación incluirá automáticamente los tabuladores cuando iniciemos una estructura condicional (lo mismo ocurrirá cuando veamos las estructuras repetitivas). En cualquier caso, si modificamos el código y añadimos/suprimimos estructuras anidadas (**if** dentro de otro **if** o **else**) podemos seleccionar el texto del código deseado y pulsar la tecla de tabulación para añadir margen o **Shift+tabulación** para quitarlo.

Vamos a emplear como base para esta práctica el ejercicio **1** (media aritmética con enteros) de la Relación de Problemas II (página **RP-II.1**).

En primer lugar, crearemos la carpeta **II_Media_int** en **U:\FP** y copiaremos en ella el fichero fuente **II_Media_int.cpp** (disponible en decsai)

Forzad los siguientes errores en tiempo de compilación, para habituarnos a los mensajes de error ofrecidos por el compilador:

- Suprime los paréntesis de alguna de las expresiones lógicas de la sentencia **if**
- Quite la llave abierta de la sentencia condicional (como únicamente hay una sentencia dentro del **if** no es necesario poner las llaves, pero añadimos las llaves a cualquier condicional para comprobar el error que se produce al eliminar una de ellas)
- Quite la llave cerrada de la sentencia condicional

Depuración

" If debugging is the process of removing bugs, then programming must be the process of putting them in. Edsger Dijkstra (1930/2002) "



Un depurador de programas (*debugger* en inglés) permite ir ejecutando un programa sentencia a sentencia (ejecución paso a paso). Además, nos permite ver en cualquier momento el valor de las variables usadas por el programa. El uso de un depurador facilita la localización de errores lógicos en nuestros programas, que de otra forma resultarían bastante difíciles de localizar directamente en el código.

"Debuggers don't remove bugs. They only show them in slow motion".



Para poder realizar tareas de depuración en Dev C++ debemos asegurarnos que estamos usando un perfil del compilador con las opciones de depuración habilitadas.

Si cuando configuramos el compilador seleccionamos **Herramientas | Opciones del Compilador | Compilador a configurar: Debug** nuestro entorno estará preparado para depurar programas.

Si no fuera así, al intentar depurar el programa, Dev C++ nos mostrará la ventana de la figura 4.

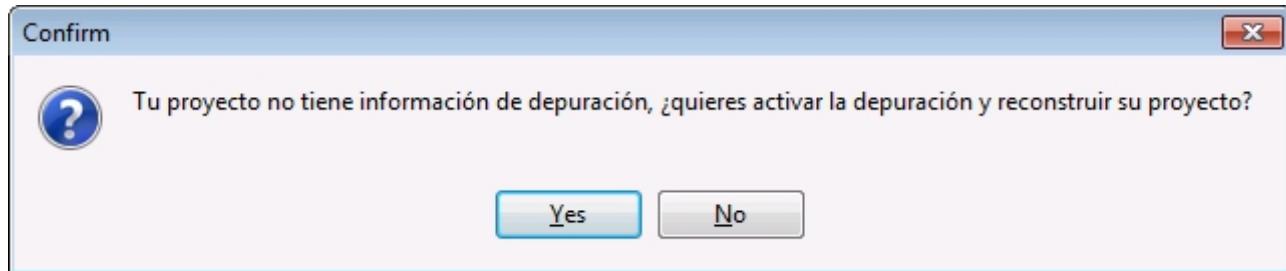


Figura 4: Ventana emergente que aparece cuando la configuración actual del compilador no permite tareas de depuración

La idea básica en la depuración es ir ejecutando el código *línea a línea* para ver posibles fallos del programa. Para eso, debemos dar los siguientes pasos:

1. Establecer una línea del programa en la que queremos que se pare la ejecución. Lo haremos introduciendo un **punto de ruptura** o (*breakpoint*) en dicha línea. Si sospechamos dónde puede estar el error, situaremos el punto de ruptura en dicha línea. En caso contrario, lo situaremos:
 - a) al principio del programa, si no sabemos exactamente dónde falla el programa, o
 - b) al principio del bloque de instrucciones del que desconfiamos, siempre y cuando tengamos confianza en todas las instrucciones que se ejecutan antes.

Para establecer un punto de ruptura podemos mover el ratón en la parte más a la izquierda de una línea de código (o sobre el número de línea) y pulsar el botón izquierdo del ratón en esa posición. La instrucción correspondiente queda marcada en rojo. Si en esa línea ya había un punto de ruptura, entonces será eliminado. También podemos colocar el cursor sobre la instrucción y con el menú contextual (botón derecho del ratón) seleccionar **Añadir/Quitar Punto de Ruptura** o simplemente, pulsar **F4**. Para eliminar un punto de ruptura, se realiza la misma operación que para incluirlo, sobre la instrucción que actualmente lo tiene.

Colocad ahora un punto de ruptura sobre la linea que contiene la primera sentencia condicional **if** (figura 5).

2. Comenzar la depuración:

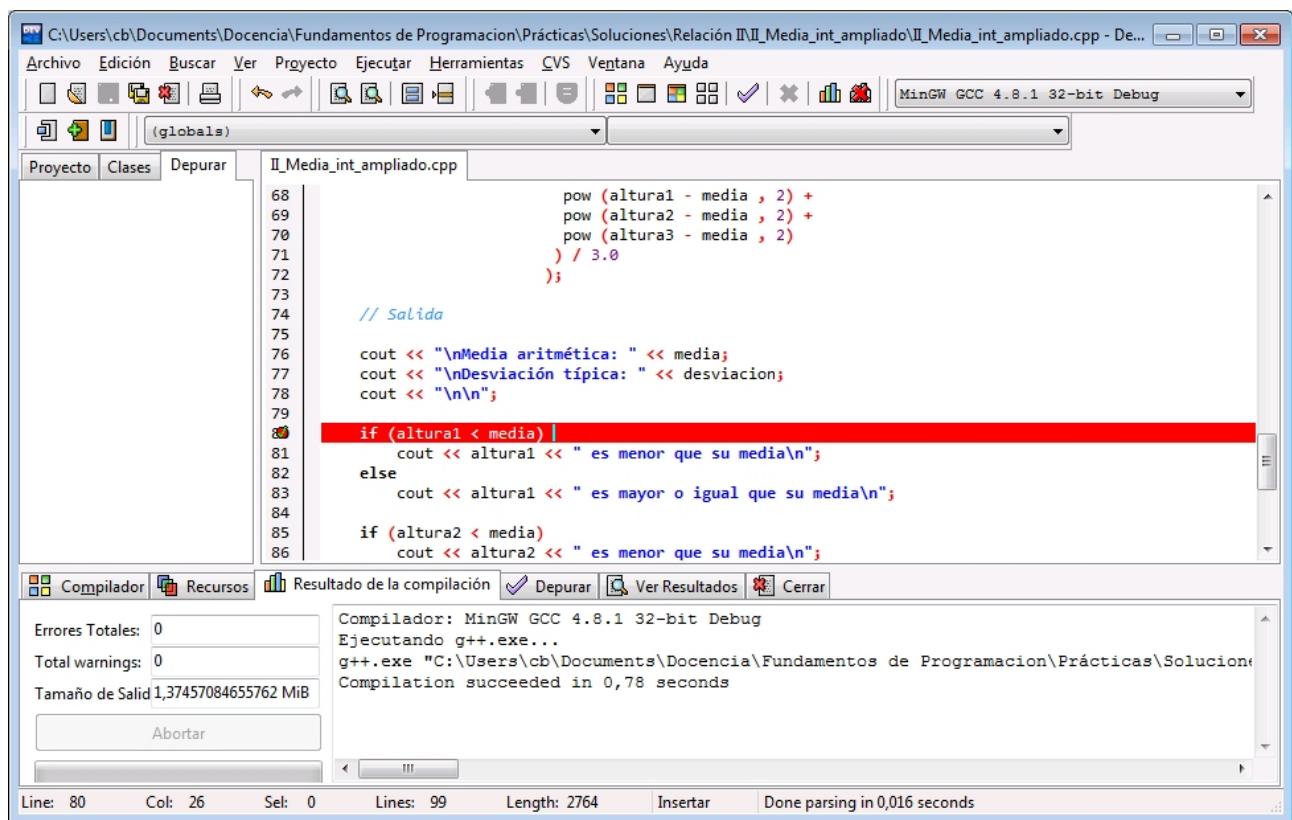


Figura 5: Se ha activado un punto de ruptura

- a) pulsar **F5**,
- b) pulsar sobre el icono 
- c) seleccionar en el menú Ejecutar | Depurar, ó
- d) en la zona inferior, pestaña Depurar, pulsar el botón **Depurar** (figura 6)

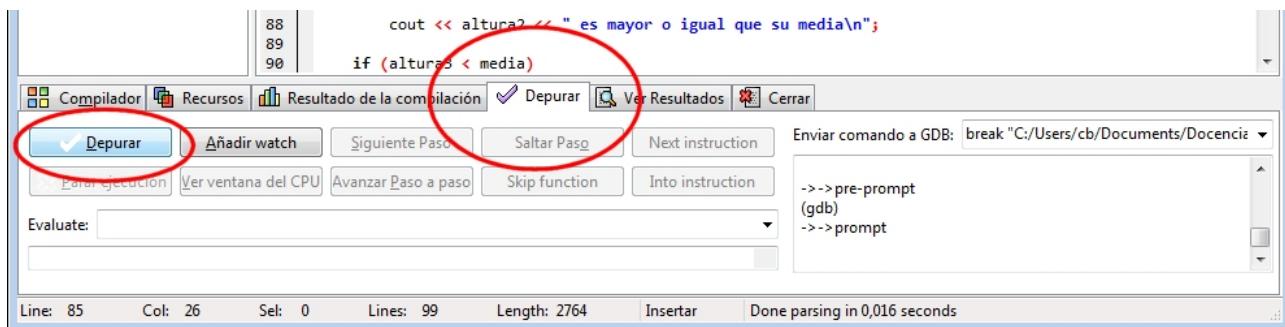


Figura 6: Inicio del proceso de depuración

Muy importante: Si se escoge *ejecutar* en lugar de *depurar*, el programa se ejecuta normalmente, sin detenerse en los puntos de ruptura.

Al iniciarse la depuración se ejecutan todas las sentencias hasta alcanzar el primer punto de ruptura. Llegado a este punto, la ejecución se interrumpe (queda “en espera”) y se muestra en azul (figura 7) la línea que se va a ejecutar a continuación (en este caso, la que contiene el punto de interrupción).

Ahora podemos escoger entre varias alternativas, todas ellas accesibles en la zona inferior (pestaña Depurar) pulsando el botón correspondiente (ver figura 7):

- **Parar ejecución**: Detener la depuración (y ejecución) del programa.
- **Siguiente Paso (F7)**: Ejecuta la siguiente instrucción. Si se trata de una llamada a una función, la ejecuta y continúa con la siguiente instrucción, sin entrar a ejecutar las instrucciones internas de la función. Las funciones se verán dentro de dos semanas.
- **Avanzar Paso a paso (F8)**: Ejecuta la siguiente instrucción. Si se trata de una llamada a una función, entra en la función y ejecuta la primera instrucción de la función, continuando la depuración dentro de la función.
- **Saltar Paso**: Ejecuta todas las instrucciones hasta encontrar un nuevo punto de ruptura, o llegar al final del programa.

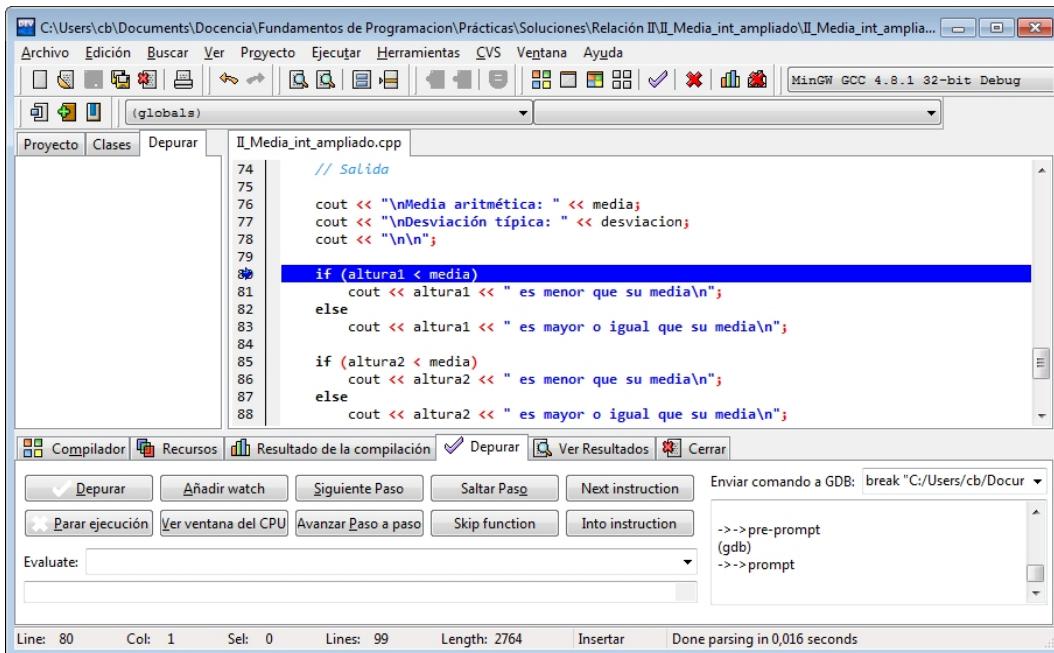


Figura 7: Inicio del proceso de depuración

La posibilidad de ver el valor de los datos que gestiona el programa durante su ejecución hace que sea más sencilla y productiva la tarea de la depuración.

La manera más sencilla de comprobar el valor que tiene una variable es colocar el cursor sobre el nombre de la variable y esperar un instante. Veremos un globo que nos muestra el nombre y valor de la variable (figura 8). El inconveniente es que al mover el ratón desaparece el globo, y cuando queramos inspeccionar nuevamente el valor de la variable debemos repetir la operación.

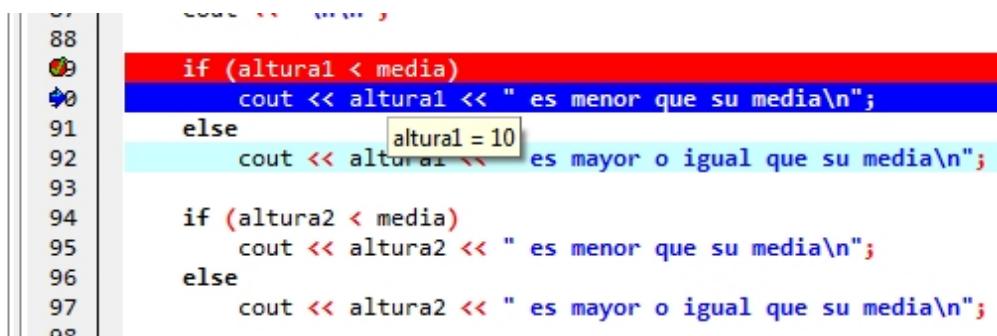
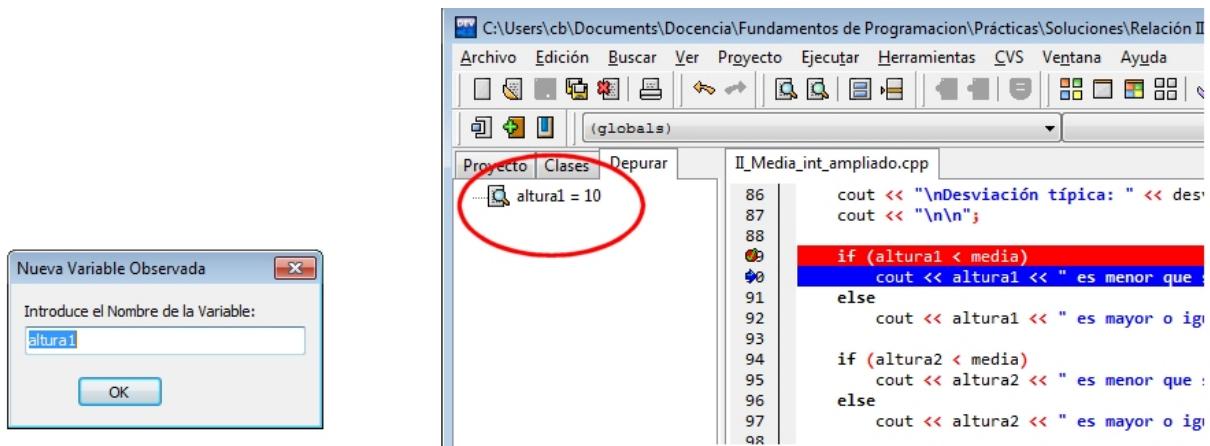


Figura 8: Inspeccionando el valor de una variable

Podemos mantener variables permanentemente monitorizadas. Aparecerán en el Explorador de Proyectos/Clases (seleccionar la pestaña Depurar).

Para añadir una variable podemos:

1. colocar el cursor sobre la variable y con el menú contextual (botón derecho del ratón) seleccionar **Añadir watch**. Aparecerá una ventana con el nombre de la variable preseleccionado (figura 9.A). Al seleccionar **OK** aparece la información de esa variable en el **Explorador de Proyectos/Clases** (figura 9.B).



A

B

Figura 9: Añadiendo una variable para su inspección permanente

2. abrir el menú contextual (botón derecho del ratón) en cualquier lugar del editor, seleccionar **Añadir watch** y escribir el nombre de la variable,
3. abrir el menú contextual en el **Explorador de Proyectos/Clases** (pestaña **Depurar**), seleccionar **Añadir watch** y escribir el nombre de la variable,
4. pulsar el botón **Añadir watch** en la zona inferior (pestaña **Depurar**) y escribir el nombre de la variable.

Conforme se ejecuta el programa podremos ver cómo cambian los valores de las variables monitorizadas.

También podríamos, incluso, modificar su valor directamente pinchando con el botón derecho sobre la variable y seleccionando **Modificar Valor**.

Otras dos opciones accesibles desde el **Explorador de Proyectos/Clases** (pestaña **Depurar**), son **Quitar watch** para eliminar una variable y **Clear All** para eliminarlas todas,

Observación final: El depurador ayuda a encontrar errores al permitir ejecutar las sentencias paso a paso y así comprobar por donde va el flujo de control y ver cómo van cambiando las variables. En cualquier caso, nunca nos cansaremos de repetir que el mejor programador es el que piensa la solución en papel, antes de escribir una sola línea de código en el entorno de programación.

"When your code does not behave as expected, do not use the debugger, think".



Resuelva el ejercicio 9 (subida salarial excluyente)

La solución de este ejercicio no hay que entregarla.

Sesión 5

Estructura Repetitiva. Bucles while

► **Actividades a realizar en casa**

Actividad: Lectura de programas resueltos.

Lea la solución de los siguientes ejercicios (disponible en decsai)

13 (Tres valores ordenados separando E/S y C con un bool)

15 (Mayúscula a minúscula y viceversa separando E/S y C con un enumerado)

19 (Divisores de un número)

Actividad: Resolución de problemas.

- *Obligatorios:*

Ejercicios sobre condicionales:

11 (Subida salarial, con condicional anidado)

14 (Año bisiesto separando E/S y C)

16 (Tres valores ordenados separando E/S y C con un enumerado)

Ejercicios sobre bucles:

22 (Gaussiana)

23 (Población)

- *Opcionales:*

24 (Leer valores dentro de un rango)

► **Actividades a realizar en las aulas de ordenadores**

Resuelva el ejercicio **29** (Mayor secuencia ascendente de temperaturas) Este ejercicio estará incluido en la próxima sesión.

Sesión 6

Actividad: Lectura de programas resueltos.

Lea la solución de los siguientes ejercicios (disponible en decsai)

33 (RLE)

38 (Narcisista)

► **Actividades a realizar en casa**

Actividad: Resolución de problemas.

Resuelva los siguientes problemas de la Relación de Problemas II.

Importante: En estos ejercicio se permiten mezclar E/S con cómputos dentro del mismo bucle (ya que todavía no se conocen herramientas para no hacerlo)

- *Obligatorios:*

28 (Lectura de los datos de la subida salarial)

29 (Mayor secuencia ascendente de temperaturas)

30 (Pinta dígitos generalizado)

- *Opcionales:*

31 (Multiplicación rusa)

► **Actividades a realizar en las aulas de ordenadores**

Empiece a trabajar con los ejercicios incluidos en la próxima sesión. Por ejemplo, puede resolver los ejercicios 40 y 46 (Factorial y sumatoria de factoriales)

Sesión 7

Estructura Repetitiva: bucles for y bucles anidados

► **Actividades a realizar en casa**

Actividad: Lectura de programas resueltos.

Lea la solución de los siguientes ejercicios (disponible en decsai)

40 (Factorial y Potencia)

42 (Parejas de caracteres)

43, 44, 45 (Triángulo y cuadrado de números)

50 (Cuenta cifras)

Actividad: Resolución de problemas.

Importante: En estos ejercicio se permiten mezclar E/S con cómputos dentro del mismo bucle (ya que todavía no se conocen herramientas para no hacerlo)

- *Obligatorios:*

De la Relación de Problemas II:

46 (Sumatoria de un factorial, bucle anidado)

49 (Número secuenciable)

De la Relación de Problemas III:

1 (Errores en funciones)

2 (Factorial y Potencia)

3 (Máximo de tres enteros)

- *Opcionales:*

47 (Sumatoria de un factorial, bucle simple)

48 (Gaussianas con un menú)

► **Actividades a realizar en las aulas de ordenadores**

Empiece a trabajar con los ejercicios incluidos en la próxima sesión.

Sesión 8

Funciones

► **Actividades a realizar en casa**

Actividad: Lectura de programas resueltos.

Lea la solución de los siguientes ejercicios (disponible en decsai)

- 4 (Lee un entero en un rango)
- 5 (Lee un entero mayor o igual que otro)
- 9 (Parking)

Actividad: Resolución de problemas.

Resuelva los siguientes ejercicios de la relación de problemas III:

- *Obligatorios:*

- 6 (Sumatoria de factoriales)
- 7 (Gaussiana)
- 8 (Gaussiana CDF)
- 10 (Población)

- *Opcionales:*

- 11 (Aumento salarial centro de atención telefónica)
- 12 (Aumento salarial centro de atención telefónica con límites variables)

Actividades de Ampliación

Familiarizarse con las webs de referencias de funciones estándar.

<http://www.cplusplus.com/reference/clibrary/>

<http://www.cppreference.com>



► **Actividades a realizar en las aulas de ordenadores**

Depuración de funciones

En la sesión 3 trabajamos sobre la depuración de programas usando Dev C++. Entonces no conocíamos cómo escribir funciones y no pudimos sacar partido a todas las opciones de depuración. En esta sesión de prácticas vamos a trabajar sobre la manera en la que se puede monitorizar la ejecución de un programa que incluye funciones.

Usaremos como ejemplo la función Combinatorio. En primer lugar, crearemos la carpeta III_Combinatorio en U:\FP y copiaremos en ella el fichero fuente III_FuncionesCombinatorio.cpp (disponible en decsai)

Antes de empezar con las tareas de depuración observaremos el *explorador de clases*. Para acceder a él, basta con seleccionar Ver | Ir al Explorador de Clases En la figura 10 puede observar que el explorador de clases muestra, para este programa, información acerca de las funciones contenidas en el fichero fuente abierto en el editor.

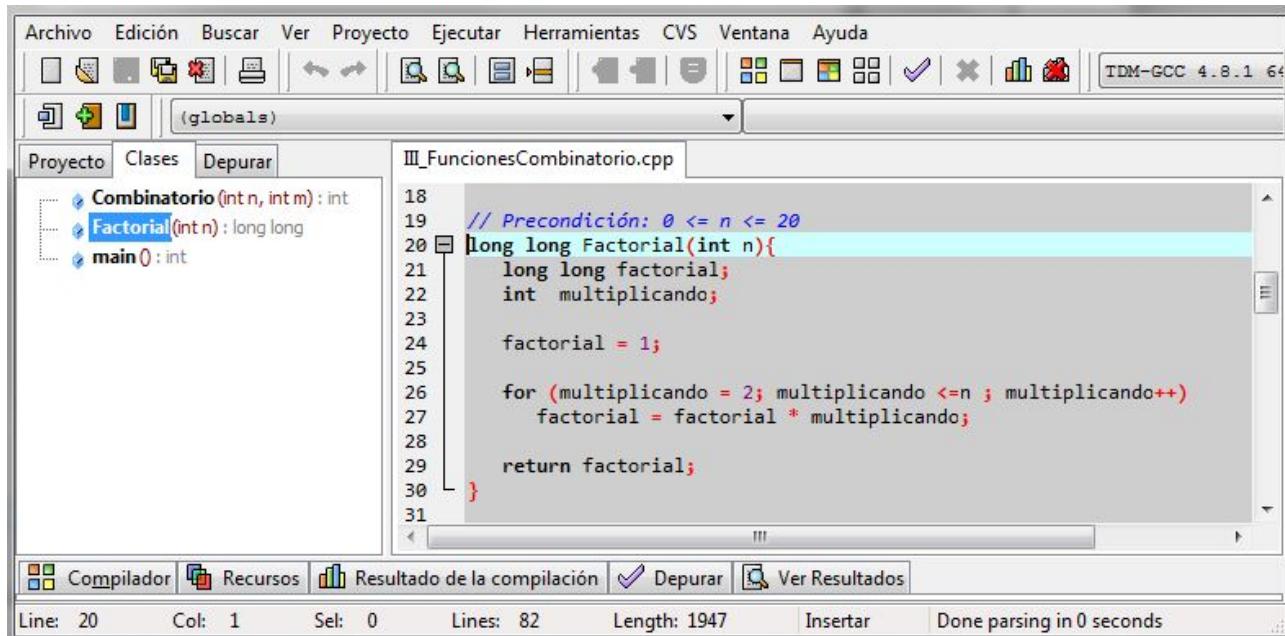


Figura 10: Explorador de clases

Para cada función muestra su *nombre*, los *parámetros formales* y su *tipo*, así como el *tipo de la función* (el tipo del valor devuelto). Se muestran todas las funciones en orden alfabético. Haciendo click sobre el nombre de cualquier función en el explorador, en el editor vemos el código de la función seleccionada. Éste es una manera rápida de acceder al código de cualquier función en nuestros programas.

El proceso de depuración se inicia de la manera habitual:

1. Fijar un punto de ruptura.
2. Comenzar la depuración.

Fijaremos un punto de ruptura, por ejemplo, en la línea del `main`

```
combinatorio = Combinatorio(total_a_elegir, elegidos);
```

y comenzamos la depuración.

El programa se ejecuta hasta llegar dicha línea, donde se detiene. Ahora podemos monitorizar su ejecución usando los botones disponibles en la zona inferior, bajo la pestaña **Depurar**.

- **Parar ejecución**: Detener la depuración (y ejecución) del programa.
- **Siguiente Paso. F7**: Ejecuta la siguiente instrucción. Si se trata de una llamada a una función, la ejecuta y continúa con la siguiente instrucción, sin entrar a ejecutar las instrucciones internas de la función.
- **Avanzar Paso a paso. F8**: Ejecuta la siguiente instrucción. Si se trata de una llamada a una función, entra en la función y ejecuta la primera instrucción de la función, continuando la depuración dentro de la función.
- **Saltar Paso**: Ejecuta todas las instrucciones hasta encontrar un nuevo punto de ruptura, o llegar al final del programa.

En la línea en la que está situado el punto de interrupción, si se pulsara **Siguiente Paso** se ejecuta completamente esa línea: la llamada a la función `Combinatorio` y la instrucción de asignación, pasando el control a la línea siguiente del `main`. Observad cómo la variable `combinatorio` se ha actualizado correctamente.

Durante la ejecución de una función pueden añadirse a la lista de variables monitorizadas cualquiera de las variables locales de la función (incluidas los parámetros formales, por supuesto). Al finalizar la ejecución de la función y dejar de estar activas las variables locales de la función veremos un mensaje de error en estas variables.

La ejecución completa de línea siguiente conlleva la ejecución de dos llamadas a la función `Factorial`:

```
denominador = Factorial(m) * Factorial(n - m);
```

En este punto,

- si se pulsa **Siguiente Paso** se completa la ejecución de esa línea y se cede el control a la línea siguiente. Observad que la variable local `combinatorio` contiene el valor ya calculado.

- si se pulsa **Avanzar Paso a paso** se entra a ejecutar la función Factorial, pasando el control a la primera instrucción de esa función.

Continúe monitorizando la ejecución del programa como desee. No se olvide de probar a establecer un punto de ruptura dentro de una función y observar qué ocurre cuando se pulsan el botón **Siguiente Paso** ¿se detiene la ejecución en el punto de ruptura o lo ignora?

Sesión 9

Clases

► ***Actividades a realizar en casa***

Actividad: Resolución de problemas.

Resuelva los siguientes ejercicios de la relación de problemas III:

- *Obligatorios:*
23 (Recta)

► ***Actividades a realizar en las aulas de ordenadores***

Esta semana se celebrará el examen de prácticas.

Sesión 10

► **Actividades a realizar en casa**

Actividad: Lectura de programas resueltos.

Lea la solución de los siguientes ejercicios (disponible en decsai)

- 24 (Cronómetro)
- 25 (Generador aleatorio de números enteros)
- 26 (Notas FP)
- 47 (Distancia euclídea con struct)

Actividad: Resolución de problemas.

Resuelva los siguientes ejercicios de la relación de problemas III:

- *Obligatorios:*

- 27 (Gaussiana con una clase)
- 28 (Población con una clase)
- 48 (Circunferencia con una clase y struct)

- *Opcionales:*

- 29 (Parking con una clase)
- 46 (Parking con una clase y struct)

► **Actividades a realizar en las aulas de ordenadores**

Emiece a trabajar con los problemas de la próxima sesión de prácticas.

Sesión 11

Vectores

► **Actividades a realizar en casa**

Actividad: Lectura de programas resueltos.

Lea la solución de los siguientes ejercicios (disponible en decsai)

1 (Palíndromo e invierte)

2 (k mayores que un valor de referencia, versión ineficiente)

Actividad: Resolución de problemas.

Resuelva los siguientes ejercicios de la relación de problemas IV:

- *Obligatorios:*

3 (k mayores que un valor de referencia, versión eficiente)

4 (Moda)

- *Opcionales:*

5 (Descodifica)

Sesión 12

► **Actividades a realizar en casa**

Actividad: Lectura de programas resueltos.

Lea la solución de los siguientes ejercicios (disponible en decsai)

8 (Palíndromo e invierte dentro de la clase SecuenciaCaracteres)

9 (Intercambia)

16 (Número de series ascendentes)

Actividad: Resolución de problemas.

Resuelva los siguientes ejercicios de la relación de problemas IV:

- *Obligatorios:*

10 (Moda dentro de la clase SecuenciaCaracteres)

11 (Elimina todas las ocurrencias de un carácter. Versión ineficiente)

12 (Elimina un bloque de caracteres. Versión ineficiente)

13 (Elimina un bloque de caracteres. Versión eficiente)

15 (Parking con un número variable de tramos)

- *Opcionales:*

14 (Elimina todas las ocurrencias de un carácter. Versión eficiente)

Sesión 13

► **Actividades a realizar en casa**

Actividad: Lectura de programas resueltos.

Lea la solución de los siguientes ejercicios (disponible en decsai)

17 (Elimina repetidos -Relación IV-)

1 (Elimina Varios -Relación V-)

Actividad: Resolución de problemas.

Resuelva los siguientes ejercicios.

- **Obligatorios:**

Resuelva los siguientes ejercicios de la relación V. Todos los algoritmos ya se han visto en sesiones anteriores. Lo único que se pide es que los encapsule en métodos dentro de una clase:

4 (Descodifica)

5 (Moda)

6 (Moda vs 2)

11 (Mayores que)

Resuelva los siguientes ejercicios de la relación V. Todos ellos son muy fáciles. No hay que diseñar ningún algoritmo complicado. Sirven para ilustrar conceptos básicos:

2 (Circunferencia)

8 (Inserta secuencia ineficiente)

9 (Inserta secuencia eficiente)

10 (Parking con la clase Instante)

15 (Circunferencia con centro un Punto2D)

Los algoritmos de los ejercicios siguientes sí son nuevos:

18 (Elimina exceso de blancos -Relación IV-)

12 (Polígono -Relación V-)

13 (Counting sort -Relación V-)

- **Opcionales:**

3 (Parking, lector tarifador)

14 (Palabra desordenada)



Fundamentos de Programación.

Relaciones de Problemas.

RELACIÓN DE PROBLEMAS I. Introducción a C++

1. Indique cuál sería el resultado de las siguientes operaciones:

```
int salario_base;
int salario_final;
int incremento;

salario_base = 1000;
salario_final = salario_base;

incremento = 200;
salario_final = salario_final + incremento;

salario_base = 3500;

cout << "\nSalario base: " << salario_base;
cout << "\nSalario final: " << salario_final;
```

Responda razonadamente a la siguiente pregunta: ¿El hecho de realizar la asignación `salario_final = salario_base;` hace que ambas variables estén ligadas durante todo el programa y que cualquier modificación posterior de `salario_base` afecte a `salario_final`?

Finalidad: Ejemplo básico de asignación a una variable del resultado de una expresión.
Dificultad Baja.

2. Cree un programa que pida un valor de intensidad y resistencia e imprima el voltaje correspondiente, según la *Ley de Ohm*:

$$\text{voltaje} = \text{intensidad} * \text{resistencia}$$

Finalidad: Ejemplo básico de asignación a una variable del resultado de una expresión.
Dificultad Baja.

3. Cree un programa que nos pida la longitud del radio y calcule el área del círculo y la longitud de la circunferencia correspondientes. Finalmente, el programa mostrará en pantalla los resultados. Recuerde que:

$$\text{área circ} = \pi r^2 \quad \text{long. circunf} = 2\pi r$$

En primera instancia, use como π el valor 3.1416. A continuación cambie el valor por 3.1415927, recompile y ejecute.

Ejemplo de entrada: 3 -- Salida correcta: 28.274 18.849

Ejemplo de entrada: 0 -- Salida correcta: 0 0

Finalidad: Resolver un problema real sencillo, usando varias sentencias. Dificultad Baja.

4. Construya un programa para leer el valor de una variable `salario_base` de tipo `double`, la incremente un 2% e imprima el resultado en pantalla. Para realizar este cálculo, multiplique por 1.02 el valor original. Para resolver este ejercicio tiene varias alternativas:
 - a) Directamente hacer el cálculo `1.02 * salario_base` dentro de la sentencia `cout`
 - b) Introducir una variable `salario_final`, asignarle la expresión anterior y mostrar su contenido en la sentencia `cout`
 - c) Modificar la variable original `salario_base` con el resultado de incrementarla un 2%.

Indique qué alternativa elige y justifíquela.

Ejemplo de entrada: 30 — Salida correcta: 30.6

Ejemplo de entrada: 0 — Salida correcta: 0

Finalidad: Ejemplo básico de asignación a una variable del resultado de una expresión. Dificultad Baja.

5. Un banco presenta la siguiente oferta. Si se deposita una cantidad de euros dada por la variable `capital` durante un año a plazo fijo, se dará un interés dado por la variable `interes`. Realice un programa que lea una cantidad `capital` y un interés `interes` desde teclado. A continuación, el programa debe calcular en una variable `total` el dinero que se tendrá al cabo de un año, aplicando la fórmula de abajo e imprimirá el resultado en pantalla.

$$\text{total} = \text{capital} + \text{capital} * \frac{\text{interes}}{100}$$

Utilice el tipo de dato `double` para todas las variables. Supondremos que el usuario introduce el interés como un valor real entre 0 y 100, es decir, un interés del 5,4 % se introducirá como 5.4. También supondremos que lo introduce correctamente, es decir, que sólo introducirá valores entre 0 y 100.

Observe que para implementar la fórmula anterior, debemos usar el operador de división que en C++ es /, por lo que nos quedaría:

```
total = capital + capital * interes / 100;
```

Es importante destacar que el compilador primero evaluará la expresión de la parte derecha de la anterior asignación (usando el valor que tuviese la variable `capital`) y a continuación ejecutará la asignación, escribiendo el valor resultante de la expresión dentro de la variable `total`.

En la asignación que calcula la variable `total`, ¿se podría sustituir dicha variable por `capital`? es decir:

```
capital = capital + capital * interes / 100;
```

Analice las ventajas o inconvenientes de hacerlo así.

Ejemplo de entrada: 300 5.4 — Salida correcta: 316.2

Ejemplo de entrada: 300 0 — Salida correcta: 300

Ejemplo de entrada: 0 5.4 — Salida correcta: 0

Finalidad: Resolver un problema real sencillo, usando varias sentencias. Dificultad Baja.

6. Calcule el número de segundos que hay entre dos instantes del mismo día.

Cada instante se caracteriza por la hora (entre 0 y 23), minuto (entre 0 y 59) y segundo (entre 0 y 59).

El programa leerá la hora, minuto y segundo del instante inicial y la hora, minuto y segundo del instante final (supondremos que los valores introducidos son correctos) y mostrará el número de segundos entre ambos instantes.

Ejemplo de entrada: 9 12 9 10 34 55 — Salida correcta: 4966

Ejemplo de entrada: 10 34 55 9 12 9 — Salida correcta: -4966

Ejemplo de entrada: 10 34 55 10 34 55 — Salida correcta: 0

Finalidad: Trabajar con expresiones numéricas y algoritmos. Dificultad Baja.

7. Queremos construir un programa que simule un juego inspirado en el de los *triles* (del que procede el nombre de *trilero*). Suponemos que hay dos participantes y cada uno tiene una caja etiquetada con su nombre. Dentro de cada caja hay una cantidad de dinero y el objetivo es intercambiar las cantidades que hay dentro. Por ahora, sólo se pide construir un programa que haga lo siguiente:

- Debe leer desde teclado dos variables `caja_izda` y `caja_dcha`.
- A continuación debe intercambiar sus valores y finalmente, mostrarlos en pantalla.

Observe que se desea intercambiar el contenido de las variables, de forma que `caja_izda` pasa a contener lo que tenía `caja_dcha` y viceversa. El siguiente código no es válido ya que simplemente engaña al usuario pero las cajas no se quedan modificadas:

```
cout << "La caja izquierda vale " << caja_dcha << "\n";
cout << "La caja derecha vale " << caja_izda;
```

Estaríamos tentados a escribir el siguiente código:

```
caja_izda = caja_dcha;
caja_dcha = caja_izda;
```

pero no funciona correctamente ¿Por qué?

Proponga una solución e impleméntela.

Finalidad: Entender cómo funciona la asignación entre variables. Dificultad Baja.

8. Realice un programa que nos pida una longitud cualquiera dada en metros. El programa deberá calcular e imprimir en pantalla el equivalente de dicha longitud en pulgadas, pies, yardas y millas. Para el cálculo, utilice la siguiente tabla de conversión del sistema métrico:

1 pulgada= 25,4 milímetros
1 pie = 30,48 centímetros
1 yarda = 0,9144 metros
1 milla = 1609,344 metros

Ejemplo de entrada: 1 — Salida correcta: 39.3701 3.28084 1.09361 0.00062

Finalidad: Plantear la solución de un ejercicio básico como es el de una conversión. Dificultad Baja.

9. Recupere la solución del ejercicio 4 (Subir sueldo usando la variable `salario_final`) Además de mostrar el salario con la subida del 2% se quiere mostrar el salario resultante de subirle otro 3% adicional. Esta segunda subida se realizará sobre el resultado de haber aplicado la primera subida. El programa debe mostrar los salarios resultantes (el resultante de la subida del 2% y el resultante de las dos subidas consecutivas del 2% y del 3%).

Ejemplo de entrada: 30 — Salida correcta: 30.6 31.518

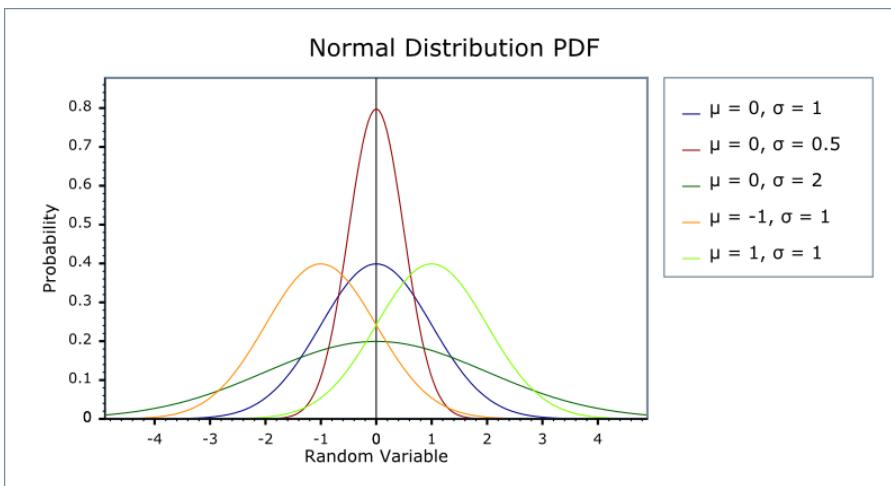
Ejemplo de entrada: 0 — Salida correcta: 0 0

Finalidad: Trabajar con expresiones numéricas y con variables para no repetir códigos. Dificultad Baja.

10. La función gaussiana es muy importante en Estadística. Es una función real de variable real que depende de dos parámetros μ y σ . El primero (μ) se conoce como *esperanza o media* y el segundo (σ) como *desviación típica* (*mean* y *standard deviation* en inglés). Su definición viene dada por la siguiente expresión:

$$\text{gaussiana}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-0,5 \left(\frac{x - \mu}{\sigma} \right)^2}$$

En la gráfica de abajo pueden verse algunos ejemplos de esta función con distintos parámetros.



Realice un programa que lea los coeficientes reales μ y σ de una función gaussiana. A continuación el programa leerá un valor de abscisa x y se imprimirá el valor que toma la función en x

Para representar el número π defina una constante con el valor 3.1416 -no use una coma para separar la parte entera de la decimal, sino un punto-

Para realizar las operaciones indicadas , debe utilizar las siguientes funciones de la biblioteca `cmath`:

- Para elevar el número e a un valor cualquiera, use la función `exp`. Por ejemplo, para calcular e^8 debería usar la siguiente expresión:

`exp(8)`

- Para calcular la raíz cuadrada, use `sqrt`.

- Para elevar un número a otro, utilice la función `pow` en la siguiente forma:

`pow(base, exponente)`

En nuestro caso, la base es $\frac{x - \mu}{\sigma}$ y el exponente 2.

Una vez resuelto el ejercicio usando la función `pow`, resuélvalo de otra forma en la que no necesite usar dicha función.

Compruebe que los resultados son correctos, usando cualquiera de las calculadoras disponibles en:

<http://danielsoper.com/statcalc3/calc.aspx?id=54>

<https://www.easycalculation.com/statistics/normal-pdf.php>

Ejemplo de entrada: 12 5 2.5 — Salida correcta: 0.01312316

Ejemplo de entrada: 0 1 0 — Salida correcta: 0.39894228

Finalidad: Trabajar con expresiones numéricas más complejas. Dificultad Media.

11. Re-escriba las soluciones de los ejercicios 3, 9, 6, 10 (circunferencia, subir sueldo, diferencia de tiempo, gaussiana) utilizando datos de tipo constante en aquellos sitios donde considere oportuno.
12. Los estudios poblacionales utilizan los conceptos de tasa de natalidad, mortalidad, etc. Al igual que un porcentaje representa una razón del total por cada cien (tanto por ciento), la tasa es una razón del total por cada mil (tanto por mil). Así pues una tasa de natalidad de 32, por ejemplo, significa que hay 32 nacimientos por cada 1000 habitantes.

Escriba un programa que calcule la estimación de la población de un territorio después de tres años. Para ello, el programa debe leer la población inicial, la tasa de natalidad, la de mortalidad y la tasa de migración. Ésta última es la diferencia entre los que se van y los que vienen, por lo que puede ser o bien positiva o bien negativa.

Suponga que todos los datos son enteros.

Tenga en cuenta que una vez calculada la población que habrá el siguiente año, las tasas se vuelven a aplicar sobre la población así obtenida, y así sucesivamente, tantos años como estemos interesados.

Ejemplo de entrada: 1375570814 32 12 7 — Salida correcta: 1490027497

*Finalidad: Ejemplo básico de asignación acumulada y uso de tipos numéricos distintos.
Dificultad Baja.*

13. Escriba un programa que lea un valor entero e imprima en pantalla cada uno de sus dígitos separados por dos espacios en blanco. Supondremos que el usuario introduce siempre un entero de tres dígitos, como por ejemplo 351. En este caso, la salida sería:

3 5 1

Finalidad: Ejemplo de asignación acumulada.

Dificultad Media.

14. Escriba un programa que calcule el consumo de gasolina. Pedirá la distancia recorrida (en kms), los litros de gasolina consumidos y los litros que quedan en el depósito. El programa debe informar el consumo en *km/litro*, los *litros/100 km* y cuántos kilómetros de autonomía le restan con ese nivel de consumo. Utilice nombres de variables significativos.

Finalidad: Resolver un problema real sencillo, usando varias sentencias. Dificultad Baja.

15. Escriba un algoritmo para calcular la media aritmética muestral y la desviación estándar (o típica) muestral de las alturas de tres personas ($n=3$). Estos valores serán

reales (de tipo `double`). La fórmula general para un valor arbitrario de `n` es:

$$\bar{X} = \frac{1}{n} \sum_{i=1}^n x_i , \quad S = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \bar{X})^2}$$

\bar{X} representa la media aritmética y S la desviación típica muestral. Para resolver este problema es necesario usar la función `sqrt` (raíz cuadrada) que se encuentra en la biblioteca `cmath`.

Estas medidas se utilizan mucho en Estadística para tener una idea de la distribución de datos. La media (mean en inglés) nos da una idea del valor central y la desviación típica (standard deviation) nos da una idea de la dispersión de éstos. Ejecutad el programa con varios valores y comprobad que el resultado es correcto utilizando una calculadora científica o cualquier calculadora online como por ejemplo la disponible en <http://www.disfrutalasmatematicas.com/datos/desviacion-estandar-calculadora.html>

Finalidad: Trabajar con expresiones numéricas y con variables para no repetir cálculos. Dificultad Baja.

16. En atletismo se expresa la rapidez de un atleta en términos de ritmo (*minutos y segundos por kilómetro*) más que en unidades de velocidad (*kilómetros por hora*).

Escribia dos programas para convertir entre estas dos medidas:

- a) El primero leerá el ritmo (minutos y segundos, por separado) y mostrará la velocidad (kilómetros por hora).
- b) El segundo leerá la velocidad (kilómetros por hora) y mostrará el ritmo (minutos y segundos).

Finalidad: Trabajar con expresiones numéricas y con variables de diferentes tipos. Dificultad Baja.

17. Las ganancias de un determinado producto se reparten entre el diseñador y los tres fabricantes del mismo. Diseñe un programa que pida la ganancia total de la empresa (los ingresos realizados con la venta del producto) y diga cuánto cobran cada uno de ellos, sabiendo que el diseñador cobra el doble que cada uno de los fabricantes. El dato de entrada será la ganancia total a repartir. Utilice el tipo `double` para todas las variables.

Importante: No repita cálculos ya realizados.

Finalidad: Entender la importancia de no repetir cálculos para evitar errores de programación. Dificultad Baja.

18. Realice el ejercicio del reparto de la ganancia de un producto, pero cambiando el tipo de dato de la ganancia total a `int` (el resto de variables siguen siendo `double`)

Finalidad: Trabajar con expresiones numéricas que involucren distintos tipos de datos.
Dificultad Baja.

19. Realice el ejercicio del cálculo de la desviación típica, pero cambiando el tipo de dato de las variables x_i a `int`.

Nota: Para no tener problemas en la llamada a la función `pow` (en el caso de que se haya utilizado para implementar el cuadrado de las diferencias de los datos con la media), obligamos a que la base de la potencia sea un real multiplicando por 1.0, por lo que la llamada quedaría en la forma `pow(base*1.0, exponente)`

Finalidad: Trabajar con expresiones numéricas que involucren distintos tipos de datos.
Dificultad Baja.

20. Realice un programa que lea una mayúscula desde teclado sobre una variable de tipo `char`. A continuación, el programa imprimirá el 0 si se ha introducido el carácter A, el 1 si era la B, el 2 si era la C y así sucesivamente. Supondremos que el usuario introduce siempre un carácter mayúscula.

Ejemplo de entrada: C — Salida correcta: 2

Finalidad: Entender el tipo de dato char. *Dificultad Baja.*

21. Supongamos el siguiente código:

```
int entero;
char caracter;

cin >> caracter;
entero = caracter;
```

Supongamos que ejecutamos el código e introducimos el 7 desde teclado. El programa está leyendo una variable de tipo `char`. Por lo tanto, el símbolo 7 se interpreta como un carácter y es como si hiciésemos la siguiente asignación:

```
caracter = '7';
entero = caracter;
```

por lo que la variable `caracter` almacenará internamente el valor 55 (el orden en la tabla ASCII del carácter `'7'`). Lo mismo ocurre con la variable `entero`, que pasa a contener 55.

Sin embargo, queremos construir un programa para asignarle a la variable `entero` el número 7 asociado al dígito representado en la variable `caracter`, es decir, el 7 y no el 55. ¿Cómo se le ocurre hacerlo? El programa también imprimirá en pantalla el resultado.

Nota. La comilla simple para representar un literal de carácter es la que hay en el teclado del ordenador debajo de la interrogación ?. Esta comilla hay que ponerla en el código pero no en la entrada del carácter desde teclado.

Ejemplo de entrada: 7 (`cin` de un `char`) —— Salida correcta: 7 (`cout` de un `int`)

Finalidad: Entender la equivalencia de C++ entre tipos enteros y de carácter. Dificultad Baja.

22. Razone sobre la falsedad o no de las siguientes afirmaciones:

- a) 'c' es una expresión de caracteres.
- b) 4 < 3 es una expresión numérica.
- c) (4 + 3) < 5 es una expresión numérica.
- d) cout << a; da como salida la escritura en pantalla de una a.
- e) ¿Qué realiza `cin >> cte`, siendo `cte` una constante entera?

Finalidad: Distinguir entre expresiones de distinto tipo de dato. Dificultad Baja.

23. Construya un programa que lea desde teclado tres variables correspondientes a un número de horas, minutos y segundos, respectivamente. A continuación, el programa debe calcular las horas, minutos y segundos dentro de su rango correspondiente. Por ejemplo, dadas 312 horas, 119 minutos y 1291 segundos, debería dar como resultado 13 días, 2 horas, 20 minutos y 31 segundos. El programa no calculará meses, años, etc. sino que se quedará en los días.

Como consejo, utilice el operador / que cuando trabaja sobre datos enteros, obtiene la división entera. Para calcular el resto de la división entera, use el operador %.

Ejemplo de entrada: 312 119 1291 —— Salida correcta: 13 2 20 31

Finalidad: Trabajar con expresiones numéricas y con variables para no repetir códigos. Dificultad Media.

24. Se quiere generalizar el ejercicio 7 que intercambiaba el valor de dos variables al caso de tres variables. Construya un programa que declare las variables x, y y z, lea su valor desde teclado e intercambien entre sí sus valores de forma que el valor de x pasa a y, el de y pasa a z y el valor de z pasa a x (se pueden declarar variables auxiliares aunque se pide que se use el menor número posible).

Ejemplo de entrada: 7 4 5 —— Salida correcta: 5 7 4

Finalidad: Mostrar la importancia en el orden de las asignaciones. Dificultad Media.

25. Construya un programa que lea un carácter (supondremos que el usuario introduce una mayúscula), lo pase a minúscula y lo imprima en pantalla. Hágalo sin usar las funciones `toupper` ni `tolower` de la biblioteca `cctype`. Para ello, debe considerarse la relación que hay en C++ entre los tipos enteros y caracteres.

Ejemplo de entrada: D —— Salida correcta: d

Finalidad: Entender la equivalencia de C++ entre tipos enteros y de carácter. Dificultad Baja.

26. Dadas las variables `count = 0`, `limit = 10`, `x = 2`, `y = 7`, calcule el valor de las siguientes expresiones lógicas

```
count == 0 && limit < 20
limit > 20 || count < 5
!(count == 12)
count == 1 && x < y
!( (count < 10 || x < y) && count >= 0 )
(count > 5 && y == 7) || (count <= 0 && limit == 5*x)
!( limit != 10 && x > y )
```

27. Escriba una expresión lógica que sea verdadera si una variable de tipo carácter llamada `letra` es una letra minúscula y falso en otro caso.

Escriba una expresión lógica que sea verdadera si una variable de tipo entero llamada `edad` es menor de 18 o mayor de 65.

Escriba una expresión lógica que sea verdadera si una variable de tipo entero llamada `adivine` está entre 1 y 100.

Escriba una expresión lógica que sea verdadera si un año es bisiesto. Los años bisiestos son aquellos que o bien son divisibles por 4 pero no por 100, o bien son divisibles por 400.

Escriba un programa que lea las variables `letra`, `edad`, `adivine` y `anio`, calcule el valor de las expresiones lógicas anteriores e imprima el resultado. Debe almacenarse el resultado de las expresiones lógicas en variables de tipo `bool`.

Tenga en cuenta que cuando se imprime por pantalla (con `cout`) una expresión lógica que es true, se imprime 1. Si es false, se imprime un 0. En el tema 2 veremos la razón.

Ejemplo de entrada: a 30 0 2017 —— Salida correcta: 1 0 0 0

Ejemplo de entrada: A 17 30 2000 —— Salida correcta: 0 1 1 1

Finalidad: Empezar a trabajar con expresiones lógicas, muy usadas en el tema 2. Dificultad Baja.

28. Indique si se produce un problema de precisión o de desbordamiento en los siguientes ejemplos y diga cuál sería el resultado final de las operaciones.

Nota. Si se desea ver el contenido de una variable real con `cout`, es necesario que antes de hacerlo, se establezca el número de decimales que se quieren mostrar en pantalla. Para ello, basta ejecutar la sentencia `cout.precision(numero_digitos);` al inicio del programa. Hay que destacar que al trabajar con reales en coma flotante (`double`, `float`, etc) siempre debemos asumir que el valor almacenado es sólo una representación aproximada.

- a)

```
int chico, chico1, chico2;
chico1 = 1234567;
chico2 = 1234567;
chico = chico1 * chico2;
```
- b)

```
long grande;
int chico1, chico2;
chico1 = 1234567;
chico2 = 1234567;
grande = chico1 * chico2;
```
- c)

```
double resultado, real1, real2;
real1 = 123.1;
real2 = 124.2;
resultado = real1 * real2;
```
- d)

```
double resultado, real1, real2;
real1 = 123456789.1;
real2 = 123456789.2;
resultado = real1 * real2;
```
- e)

```
double real, otro_real;
real = 2e34;
otro_real = real + 1;
otro_real = otro_real - real;
```
- f)

```
double real, otro_real;
real = 1e+300;
otro_real = 1e+200;
otro_real = otro_real * real;
```
- g)

```
float chico;
double grande;
grande = 2e+150;
chico = grande;
```

Finalidad: Entender los problemas de desbordamiento y precisión. Dificultad Media.

29. Indique qué tipo de dato usaría para representar:

- Edad de una persona
- Producto interior bruto de un país. Consultad:
[http://es.wikipedia.org/wiki/Anexo:Pa%C3%ADses_por_PIB_\(nominal\)](http://es.wikipedia.org/wiki/Anexo:Pa%C3%ADses_por_PIB_(nominal))
- La cualidad de que un número entero sea primo o no.
- Estado civil (casado, soltero, separado, viudo)
- Sexo de una persona (hombre o mujer exclusivamente)

Finalidad: Saber elegir adecuadamente un tipo de dato, atendiendo a la información que se quiere representar. Dificultad Media.

30. El precio final de un automóvil para un comprador es la suma total del costo del vehículo, del porcentaje de ganancia de dicho vendedor y del I.V.A. Diseñe un algoritmo para obtener el precio final de un automóvil sabiendo que el porcentaje de ganancia de este vendedor es del 20 % y el I.V.A. aplicable es del 16 %.

Dificultad Baja.

31. Cree un programa que lea un valor de temperatura expresada en grados Celsius y la transforme en grados Fahrenheit. Para ello, debe considerar la fórmula siguiente:

$$\text{Grados Fahrenheit} = (\text{Grados Celsius} * 180 / 100) + 32$$

Buscad en Internet el por qué de dicha fórmula.

Dificultad Baja.

32. Cree un programa que lea las coordenadas de dos puntos $P_1 = (x_1, y_1)$ y $P_2 = (x_2, y_2)$ y calcule la distancia euclídea entre ellos:

$$d(P_1, P_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Para calcular el cuadrado no puede usar ninguna función de la biblioteca `cmath`.

33. Declare las variables necesarias y traduzca las siguientes fórmulas a expresiones válidas del lenguaje C++.

$$a) \frac{1 + \frac{x^2}{y}}{\frac{x^3}{1+y}}$$

$$b) \frac{1 + \frac{1}{3} \sin h - \frac{1}{7} \cos h}{2 h}$$

$$c) \sqrt{1 + \left(\frac{e^x}{x^2}\right)^2}$$

Algunas funciones de `cmath`

<code>sen(x)</code>	→ <code>sin(x)</code>
<code>cos(x)</code>	→ <code>cos(x)</code>
<code>x^y</code>	→ <code>pow(x, y)</code>
<code>ln(x)</code>	→ <code>log(x)</code>
<code>e^x</code>	→ <code>exp(x)</code>

Dificultad Baja.

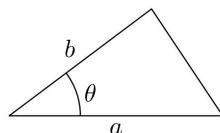
34. Dos locomotoras parten de puntos distintos avanzando en dirección contraria sobre la misma vía. Se pide redactar un programa para conocer las distancias que habrán recorrido ambas locomotoras antes de que choquen teniendo en cuenta que la primera locomotora viaja a una velocidad constante V_1 , que la segunda viaja a una velocidad constante V_2 , la fórmula que relaciona velocidad, espacio y tiempo ($s = v t$) y que el momento en que se producirá el choque viene dado por la fórmula

$$t = \frac{D}{V_1 + V_2}$$

dónde D es la distancia que separa los puntos iniciales de partida. Los datos de entrada al programa serán D , V_1 y V_2 .

Dificultad Baja.

35. El área A de un triángulo se puede calcular a partir del valor de dos de sus lados, a y b , y del ángulo θ que éstos forman entre sí con la fórmula $A = \frac{1}{2}ab \sin(\theta)$. Construya un programa que pida al usuario el valor de los dos lados (en centímetros), el ángulo que éstos forman (en grados), y muestre el valor del área.



Tened en cuenta que el argumento de la función `sin` va en radianes por lo que habrá que transformar los grados del ángulo en radianes (recordad que 360 grados son 2π radianes).

Dificultad Baja.

36. Los compiladores utilizan siempre el mismo número de bits para representar un tipo de dato entero (este número puede variar de un compilador a otro). Por ejemplo, 32 bits para un `int`. Pero, realmente, no se necesitan 32 bits para representar el 6, por ejemplo, ya que bastarían 3 bits:

$$6 = 1 * 2^2 + 1 * 2^1 + 0 * 2^0 \equiv 110$$

Se pide crear un programa que lea un entero n , y calcule el mínimo número de dígitos que se necesitan para su representación. Para simplificar los cálculos, suponed que sólo queremos representar valores enteros positivos (incluido el cero). Consejo: se necesitará usar el logaritmo en base 2 y obtener la parte entera de un real (se obtiene tras el truncamiento que se produce al asignar un real a un entero)

Dificultad Media.

RELACIÓN DE PROBLEMAS II. Estructuras de Control

Ejercicios sobre condicionales

1. Amplie el ejercicio 15 de la relación de problemas I, para que, una vez calculada la media y la desviación, el programa imprima por cada uno de los valores introducidos previamente, si está por encima o por debajo de la media. Por ejemplo:

33 es menor que su media
48 es mayor o igual que su media
.....

Nota. Los valores introducidos son enteros, pero la media y la desviación son reales.

Finalidad: Plantear un ejemplo básico con varias estructuras condicionales dobles consecutivas. Dificultad Baja.

2. Se quiere leer un carácter `letra_original` desde teclado, y comprobar con una estructura condicional si es una letra mayúscula. En dicho caso, hay que calcular la minúscula correspondiente almacenando el resultado en una variable llamada `letra_convertida`. En el caso de que no sea una mayúscula, le asignaremos a `letra_convertida` el valor que tenga `letra_original`. Finalmente, imprimiremos en pantalla el valor de `letra_convertida`. No pueden usarse las funciones `tolower` ni `toupper` de la biblioteca `cctype`.

Ejemplo de entrada: D — Salida correcta: d

Ejemplo de entrada: d — Salida correcta: d

Ejemplo de entrada: ! — Salida correcta: !

Finalidad: Plantear una estructura condicional doble con una expresión lógica compuesta. Dificultad Baja.

3. Realice un programa en C++ que lea dos valores enteros desde teclado y diga si cualquiera de ellos divide o no (de forma entera) al otro. En este problema no hace falta decir quién divide a quién. Supondremos que los valores leídos desde teclado son ambos distintos de cero.

Finalidad: Plantear una estructura condicional doble con una expresión lógica compuesta. Dificultad Baja.

4. Queremos modificar el ejercicio 2 para leer un carácter `letra_original` desde teclado y hacer lo siguiente:

- Si es una letra mayúscula, almacenaremos en la variable `letra_convertida` la correspondiente letra minúscula.

- Si es una letra minúscula, almacenaremos en la variable `letra_convertida` la correspondiente letra mayúscula.
- Si es un carácter no alfabético, almacenaremos el mismo carácter en la variable `letra_convertida`

El programa debe imprimir en pantalla el valor de `letra_convertida` e indicar si la letra introducida era una minúscula, mayúscula o no era una carácter alfabético. No pueden usarse las funciones `tolower` ni `toupper` de la biblioteca `cctype`.

Finalidad: Plantear una estructura condicional anidada. Dificultad Baja.

- Queremos gestionar la nómina de los empleados de un centro de atención telefónica. Construya un programa que lea el salario por hora (dato de tipo real) de un empleado, el número de horas trabajadas durante el mes actual (dato de tipo entero) el número de casos resueltos de forma satisfactoria (dato de tipo entero) y el grado medio de satisfacción de los usuarios de los servicios telefónicos con el empleado en cuestión (real entre 0 y 5).

Se quiere aplicar una subida salarial en función de varios factores. En ejercicios sucesivos se irán planteando distintas posibilidades. La primera que se quiere implementar es la siguiente:

Se aplicará una subida del 4% a los empleados que han resuelto más de 30 casos.

Más de 30 casos resueltos:	+4%
----------------------------	-----

Imprima el salario final en pantalla.

Ejemplo de entrada: 8.5 150 32 5 -- Salida correcta: 1326

Ejemplo de entrada: 7.5 130 24 3 -- Salida correcta: 975

Finalidad: Plantear una estructura condicional de actualización de una variable. Dificultad Baja.

- Recupere la solución del ejercicio 5 sobre el cómputo de la nómina de los trabajadores de un centro de atención telefónica. Implemente ahora el siguiente criterio para la subida salarial. Se aplicará una subida del 4% a los empleados que han resuelto más de 30 casos y una subida del 2% si el grado de satisfacción media de los usuarios es mayor o igual que 4.0. Ambas subidas son compatibles, es decir, si un trabajador cumple las dos condiciones, se le aplicarán ambas subidas.

Resuelva este ejercicio considerando que la nueva subida del 2% se realiza sobre el salario inicial y no sobre el resultado de haber aplicado, en su caso, la otra subida del 4%.

Más de 30 casos resueltos:	+4%
Grado de satisfacción >= 4:	+2%

Ejemplo de entrada: 8.5 150 32 5 -- Salida correcta: 1351.5

Ejemplo de entrada: 8.5 150 29 5 -- Salida correcta: 1300.5

Ejemplo de entrada: 7.5 130 24 3 -- Salida correcta: 975

Finalidad: Plantear estructuras condicionales consecutivas. Dificultad Baja.

7. Escriba un programa en C++ para que lea tres enteros desde teclado y nos diga si están ordenados (da igual si es de forma ascendente o descendente) o no lo están. Por ejemplo, la sucesión de números 3, 6, 9 estaría ordenada así como la serie 13, 2, 1 pero no lo estaría la serie 3, 9, 5.

Finalidad: Plantear una estructura condicional doble con una expresión lógica compuesta. Dificultad Baja.

8. Cree un programa que lea el número de un año e indique si es bisiesto o no. Un año es bisiesto si es múltiplo de cuatro, pero no de cien. Excepción a la regla anterior son los múltiplos de cuatrocientos que siempre son bisiestos. Por ejemplo, son bisiestos: 1600, 1996, 2000, 2004. No son bisiestos: 1700, 1800, 1900, 1998, 2002.

Finalidad: Plantear una estructura condicional doble con una expresión lógica compuesta. Dificultad Baja.

9. Modifique la solución del ejercicio 6 para que ambas subidas salariales sean excluyentes, es decir, si se aplica una, no se aplicará la otra. En el caso de que ambas sean aplicables, debe aplicarse la subida más ventajosa para el trabajador, es decir, la del 4%.

De forma exclusiva:

Más de 30 casos resueltos:	+4%
Grado de satisfacción >= 4:	+2%

Ejemplo de entrada: 8.5 150 32 5 -- Salida correcta:

Ejemplo de entrada: 8.5 150 29 5 -- Salida correcta:

Ejemplo de entrada: 7.5 130 24 3 -- Salida correcta:

Finalidad: Plantear estructuras condicionales anidadas. Dificultad Baja.

10. La tabla para el cálculo del precio a pagar en los parkings de Madrid para el 2015 es la siguiente:

Si permanece más de 660 minutos se paga una única tarifa de 31.55 euros
Desde el minuto 0 al 30: 0.0412 euros cada minuto
Desde el minuto 31 al 90: 0.0370 euros cada minuto
Desde el minuto 91 al 120: 0.0311 euros cada minuto
Desde el minuto 121 al 660: 0.0305 euros cada minuto

Dado un tiempo de entrada (hora, minuto y segundo) y un tiempo de salida, construya un programa que calcule la tarifa final a cobrar. Para calcular el número de minutos entre los dos instantes de tiempo, puede utilizar la solución del ejercicio 6 de la Relación de Problemas I.

Ejemplo de entrada: 2 1 30 2 1 29 — Salida correcta: -1
Ejemplo de entrada: 2 1 30 2 1 31 — Salida correcta: 0
Ejemplo de entrada: 2 1 30 2 2 31 — Salida correcta: 0.0412
Ejemplo de entrada: 2 1 30 2 41 31 — Salida correcta: 1.606
Ejemplo de entrada: 2 1 30 3 41 31 — Salida correcta: 3.767
Ejemplo de entrada: 2 1 30 5 41 31 — Salida correcta: 7.439
Ejemplo de entrada: 2 1 30 23 1 1 — Salida correcta: 31.55

Finalidad: Plantear una estructura condicional anidada. Dificultad Baja.

11. Modifique la solución del ejercicio 6 para que también aplique una subida del 3% a los que han resuelto entre 20 y 30 casos:

Entre 20 y 30 casos resueltos:	+3%
Más de 30 casos resueltos:	+4%

Grado de satisfacción >= 4:	+2%
-----------------------------	-----

Ejemplo de entrada: 8.5 150 32 5 — Salida correcta: 1351.5
Ejemplo de entrada: 7.5 130 24 3 — Salida correcta: 1004.25
Ejemplo de entrada: 7.5 130 24 4 — Salida correcta: 1023.75

Finalidad: Plantear una estructura condicional anidada. Dificultad Baja.

12. Construya un programa para calcular el importe total a facturar de un pedido. El programa leerá el número de unidades vendidas y el precio de venta de cada unidad. Si la cantidad vendida es mayor de 100 unidades, se le aplica un descuento del 3 %. Por otra parte, si el precio final de la venta es mayor de 700 euros, se aplica un descuento del 2 %. Ambos descuentos son acumulables. Obtenga el importe final e imprimalo en pantalla.

Vamos a cambiar el criterio de los descuentos. Supondremos que sólo se aplicará el descuento del 2 % (por una venta mayor de 700 euros) cuando se hayan vendido más de 100 unidades, es decir, para ventas de menos de 100 unidades no se aplica el descuento del 2 % aunque el importe sea mayor de 700 euros.

Cambiar el programa para incorporar este nuevo criterio.

Finalidad: Plantear una estructura condicional anidada. Dificultad Baja.

13. Modifique la solución del ejercicio 7 (valores ordenados) para que no se mezclen E/S y C (entradas/salidas y cómputos) dentro de la misma estructura condicional.

Finalidad: Diseñar programas que separen Entradas/Salidas y cómputos. Dificultad Baja.

RELACIÓN DE PROBLEMAS II. Estructuras de Control

14. Modifique la solución del ejercicio 8 (año bisiesto) para que no se mezclen E/S y C (entradas/salidas y cómputos) dentro de la misma estructura condicional.

Finalidad: Diseñar programas que separen Entradas/Salidas y cómputos. Dificultad Baja.

15. Modifique la solución al ejercicio 4 para que, dependiendo de cómo era la letra introducida, imprima en pantalla alguno de los siguientes mensajes:

- La letra era una mayúscula. Una vez convertida es ...
- La letra era una minúscula. Una vez convertida es ...
- El carácter no era una letra.

Hágalo separando entradas y salidas de los cómputos. Para ello, utilice una variable de tipo enumerado que represente las opciones de que un carácter sea una mayúscula, una minúscula o un carácter no alfabético.

Finalidad: Separar E/S y C. Usar el tipo enumerado para detectar cuándo se produce una situación determinada. Dificultad Media.

16. Modifique el ejercicio 7 para que el programa nos diga si los tres valores leídos están ordenados de forma ascendente, ordenados de forma descendente o no están ordenados. Para resolver este problema, debe usar una variable de tipo enumerado.

Finalidad: Separar E/S y C. Usar el tipo enumerado para detectar cuándo se produce una situación determinada. Dificultad Baja.

17. Cree un programa que lea los datos fiscales de una persona, reajuste su renta bruta según el criterio que se indica posteriormente e imprima su renta neta final.

- La renta bruta es la cantidad de dinero íntegra que el trabajador gana.
- La retención fiscal es el tanto por ciento que el gobierno se queda.
- La renta neta es la cantidad que le queda al trabajador después de quitarle el porcentaje de retención fiscal, es decir:

$$\text{Renta_neta} = \text{Renta_bruta} - \text{Renta_bruta} * \text{Retención final} / 100$$

Los datos a leer son:

- Si la persona es un trabajador autónomo o no
- Si es pensionista o no
- Estado civil
- Renta bruta (total de ingresos obtenidos)
- Retención inicial a aplicar.

La retención inicial se va a modificar ahora atendiendo al siguiente criterio:

- Se baja 3 puntos la retención fiscal a los autónomos, es decir, si la retención inicial era de un 15 %, por ejemplo, la retención final a aplicar será de un 12 % (por lo que la renta neta final será mayor)
- Para los no autónomos:
 - Se sube un punto la retención fiscal a todos los pensionistas, es decir, si la retención inicial era de un 13 %, por ejemplo, la retención final a aplicar será de un 14 % (por lo que la renta neta final será menor)
 - Al resto de trabajadores (no autónomo y no pensionista) se le aplica a todos una primera subida lineal de dos puntos en la retención inicial.
Una vez hecha esta subida, se le aplica (sobre el resultado anterior) las siguientes subidas **adicionales**, dependiendo de su estado civil y niveles de ingresos:
 - Se sube otros dos puntos la retención fiscal si la renta bruta es menor de 20.000 euros
 - Se sube otros 2.5 puntos la retención fiscal a los casados con renta bruta superior a 20.000 euros
 - Se sube otros tres puntos la retención fiscal a los solteros con renta bruta superior a 20.000 euros

Una vez calculada la retención final, habrá que aplicarla sobre la renta bruta para así obtener la renta final del trabajador.

Finalidad: Plantear una estructura condicional anidada. Dificultad Media.

18. Una compañía aérea establece el precio del billete como sigue: en primer lugar se fija una tarifa base de 150 euros, la misma para todos los destinos. Si el destino está a menos de 200 kilómetros, el precio final es la tarifa inicial. Para destinos a más de 200 Km, se suman 10 céntimos por cada kilómetro de distancia al destino (a partir del Km 200). En una campaña de promoción se va a realizar una rebaja lineal de 15 euros a todos los viajes. Además, se pretenden añadir otras rebajas y se barajan las siguientes alternativas de políticas de descuento:
- a) Una rebaja del 3 % en el precio final, para destinos a más de 600Km.
 - b) Una rebaja del 4 % en el precio final, para destinos a más de 1100Km. En este caso, no se aplica el anterior descuento.
 - c) Una rebaja del 5 % si el comprador es cliente previo de la empresa.

Cree un programa para que lea el número de kilómetros al destino y si el billete corresponde a un cliente previo de la empresa. Calcular el precio final del billete con las siguientes políticas de descuento:

- Aplicando c) de forma adicional a los descuentos a) y b)

RELACIÓN DE PROBLEMAS II. Estructuras de Control

- Aplicando c) de forma exclusiva con los anteriores, es decir, que si se aplica c), no se aplicaría ni a) ni b)

Finalidad: Plantear una estructura condicional anidada. Dificultad Media.

Ejercicios sobre bucles

19. Realice un programa que lea desde teclado un entero **tope** e imprima en pantalla todos sus divisores propios. Para obtener los divisores, basta recorrer todos los enteros menores que el valor introducido y comprobar si lo dividen. A continuación, mejorar el ejercicio obligando al usuario a introducir un entero positivo, usando un filtro con un bucle post test (`do while`).

Finalidad: Plantear un ejemplo sencillo de bucle y de filtro de entrada de datos. Dificultad Baja.

20. Modifiquemos el ejercicio 5 del capital y los intereses de la primera relación. Supongamos ahora que se quiere reinvertir todo el dinero obtenido (el original C más los intereses producidos) en otro plazo fijo a un año y así, sucesivamente. Construya un programa para que lea el capital, el interés y un número de años N , y calcule e imprima todo el dinero obtenido durante cada uno de los N años, suponiendo que todo lo ganado (incluido el capital original C) se reinvierte a plazo fijo durante el siguiente año. El programa debe mostrar una salida del tipo:

Total en el año número 1 = 240
Total en el año número 2 = 288
Total en el año número 3 = 345.6
.....

Finalidad: Usar una variable acumuladora dentro del cuerpo de un bucle (aparecerá a la izquierda y a la derecha de una asignación). Dificultad Baja.

21. Sobre el mismo ejercicio del capital y los intereses, construya un programa para calcular cuántos años han de pasar hasta llegar a doblar, como mínimo, el capital inicial. Los datos que han de leerse desde teclado son el capital inicial y el interés anual.

Finalidad: Usar la variable acumuladora en la misma condición del bucle. Dificultad Baja.

22. Recupere la solución del ejercicio 10 (función gaussiana) de la relación de problemas I. Se pide construir un programa para imprimir el resultado de aplicar dicha función a varios valores de abscisas.

En primer lugar, se leerán los parámetros que definen la función, es decir, la esperanza y la desviación. La esperanza puede ser cualquier valor, pero para leer el valor de desviación debe utilizar un bucle y obligar a que sea mayor o igual que cero.

A continuación el programa pedirá un valor **minimo**, un valor **maximo** y un **incremento**. El valor **maximo** ha de leerse con un filtro de entrada obligando a que sea mayor que **minimo**. El programa mostrará el valor de la función gaussiana en todos los valores de x (la abscisa) entre **minimo** y **maximo** a saltos de **incremento**,

es decir, `minimo`, `minimo + incremento`, `minimo + 2*incremento`, ..., hasta llegar, como mucho, a `maximo`.

```
12      <- Media
5       <- Desviación
2       <- Mínimo
3       <- Máximo
0.5    <- Incremento
```

Ejemplo de entrada: 12 5 2 3 0.5

-- Salida correcta: 0.0107982 0.0131232 0.01579

Ejemplo de entrada: 12 -5 5 2 3 0.5

-- Salida correcta: 0.0107982 0.0131232 0.01579

Ejemplo de entrada: 12 -5 5 2 1 0 3 0.5

-- Salida correcta: 0.0107982 0.0131232 0.01579

Finalidad: Ejemplo básico de bucle. Dificultad Baja.

23. Amplie el ejercicio 12 (Población) de la relación de problemas I.

Esta nueva versión del programa, además de los datos ya pedidos en dicho ejercicio, se le pedirá al usuario que introduzca un número de años (será el último dato leído). Debe leer cada dato con un filtro conveniente. Por ejemplo, las tasas de natalidad, mortalidad y emigración deben ser enteros entre 0 y 1000, mientras que la población inicial debe ser un entero positivo.

El programa debe calcular e imprimir el número total de habitantes transcurridos dichos años.

Además, el programa también calculará el número de años que tienen que pasar hasta que haya, como mínimo, el doble de la población inicial. Imprima dicho número de años, junto con la población que habrá pasado ese tiempo.

Por ejemplo, para la siguiente entrada

```
1375570814  <- Población inicial
32           <- Tasa de natalidad
12           <- Tasa de mortalidad
7            <- Tasa de migración
3            <- Número de años
```

el programa debe devolver lo siguiente:

```
1490027497  <- Número de habitantes pasados 3 años
27           <- Años que han de pasar hasta doblar la población
2824131580  <- Población transcurridos 27 años
```

Ejemplo de entrada: 1375570814 32 12 7 3

-- Salida correcta: 1490027497 27 2824131580

Finalidad: Ejemplo básico de asignación acumulada. Dificultad Baja.

24. Se pide leer dos enteros `min` y `max` que representarán un rango de valores `[min, max]`. El primer valor a leer, `min`, debe ser un número positivo y el segundo valor `max`, debe ser mayor que `min`. El programa irá leyendo estos dos valores hasta que el usuario los introduzca correctamente.

Una vez leídos ambos valores, el programa pedirá otro entero `nuevo` obligando a que esté dentro del intervalo `[min, max]`. Si el usuario introduce más de 3 valores fuera del rango, el bucle terminará y se mostrará en pantalla un mensaje indicando que superó el número de intentos máximo. En caso contrario, es decir, el usuario introduce un valor en el rango pedido, el bucle también terminará y se mostrará en pantalla el resultado de calcular `nuevo - min + max - nuevo`.

Ejemplo de entrada: -5 -6 -7 -1 5 3 4 2 8 7

-- Salida correcta: 2 1

Ejemplo de entrada: -5 -6 -7 -1 5 3 4 2 8 4 9 7

-- Salida correcta: 2 1

Ejemplo de entrada: -5 -6 -7 -1 5 3 4 2 8 4 9 10

-- Salida correcta: Número de intentos sobrepasado

Finalidad: Trabajar con bucles con condiciones compuestas en filtros de entrada de datos. Dificultad Media.

25. Se pide leer un carácter desde teclado, obligando al usuario a que sea una letra mayúscula. Para ello, habrá que usar una estructura repetitiva `do while`, de forma que si el usuario introduce un carácter que no sea una letra mayúscula, se le volverá a pedir otro carácter. Calcule la minúscula correspondiente e imprimala en pantalla. No pueden usarse las funciones `tolower` ni `toupper` de la biblioteca `cctype`.

Finalidad: Trabajar con bucles con condiciones compuestas. Dificultad Baja.

26. Realice un programa que lea enteros desde teclado y calcule cuántos se han introducido y cual es el mínimo de dichos valores (pueden ser positivos o negativos). Se dejará de leer datos cuando el usuario introduzca el valor 0. Realice la lectura de los enteros dentro de un bucle sobre una única variable llamada `dato`. Es importante controlar los casos extremos, como por ejemplo, que el primer valor leído fuese ya el terminador de entrada (en este caso, el cero).

Finalidad: Destacar la importancia de las inicializaciones antes de entrar al bucle. Ejemplo de lectura anticipada. Dificultad Baja.

27. Amplíe el ejercicio 6 de manera que se permita que los dos instantes puedan pertenecer a dos días distintos, pero eso sí, consecutivos.

Filtrar adecuadamente los datos leídos.

Finalidad: Trabajar con condicionales complejos y filtros de entradas de datos. Reutilizar código ya escrito y verificado. Dificultad Media.

28. Se quiere construir un programa para leer los datos necesarios del ejercicio 11 de la subida salarial.

Supondremos que sólo hay tres empleados y que están identificados con un código (1, 2 y 3). Además, el salario por hora es el mismo para todos los empleados. Éste será el primer valor que se leerá (de tipo double). Después de haber leído este dato, se leerán los datos de los casos atendidos por los empleados en el siguiente orden: en primer lugar, el código del empleado, a continuación el número de segundos que ha durado la atención telefónica, en tercer lugar un 1 si el caso se resolvió de forma satisfactoria y un 0 en caso contrario; finalmente, un valor entero entre 0 y 5 con el grado de satisfacción del usuario.

Cuando nos encontremos el terminador -1 como primer dato (código del empleado) se detendrá la introducción de datos. Supondremos que siempre se introduce al menos el primer valor (el salario), pudiendo ser ya el siguiente dato leído el terminador.

```
7.5      <- Salario de 7.5 euros por hora
2 124 1 3 <- Empleado 2, 124'', resuelto, grado sat: 3
1 32 0 0  <- Empleado 1, 32'', no resuelto, grado sat: 0
2 26 0 2  <- Empleado 2, 26'', no resuelto, grado sat: 2
-1        <- Fin de entrada de datos
```

El programa debe imprimir el número total de casos introducidos (3 en el ejemplo anterior) y el código del empleado con mayor grado de satisfacción medio (también imprimirá dicho grado medio). En el ejemplo anterior, sería el empleado 2 con un nivel medio de satisfacción de 2.5 (observe que el grado medio se calcula en relación al número total de casos atendidos y no sobre los casos resueltos).

Observe que, en este ejercicio, no se están teniendo en cuenta los datos referentes al tiempo de cada caso y si fue resuelto o no, pero hay que leer todos los datos para llegar a los que sí nos interesan.

Ejemplo de entrada: 7.5 2 124 1 3 1 32 0 0 2 26 0 2 -1
-- Salida correcta: 3 2 2.5

Ejemplo de entrada: 7.5 -1

-- Salida correcta: No se introdujo ningún caso

Finalidad: Plantear un bucle de lectura de datos. Dificultad Baja.

29. Construya un programa que calcule cuándo se produjo la mayor secuencia de días consecutivos con temperaturas crecientes. El programa leerá una secuencia de reales representando temperaturas, hasta llegar al -1 y debe calcular la subsecuencia de números ordenada, de menor a mayor, de mayor longitud.

El programa nos debe decir la posición donde comienza la subsecuencia y su longitud. Por ejemplo, ante la entrada siguiente:

```
17.2 17.3 16.2 16.4 17.1 19.2 18.9 -1.0
```

el programa nos debe indicar que la mayor subsecuencia empieza en la posición 3 (en el 16.2) y tiene longitud 4 (termina en 19.2)

Puede suponer que siempre se introducirá al menos un valor de temperatura.

Ejemplo de entrada: 17.2 17.3 16.2 16.4 17.1 19.2 18.9 -1

-- Salida correcta: 3 4

Ejemplo de entrada: 17.2 17.3 16.2 16.4 17.1 19.2 -1

-- Salida correcta: 3 4

Ejemplo de entrada: 17.2 17.3 -1 -- Salida correcta: 1 2

Ejemplo de entrada: 17.2 15.3 -1 -- Salida correcta: 1 1

Ejemplo de entrada: 17.2 -1 -- Salida correcta: 1 1

Finalidad: Trabajar con bucles que comparan un valor actual con otro anterior. Dificultad Media.

30. En el ejercicio 13 de la Relación de Problemas I se pedía escribir un programa que leyese un valor entero de tres dígitos e imprimiese los dígitos separados por un espacio en blanco. Haga lo mismo pero para un número entero arbitrario. Por ejemplo, si el número es 3519, la salida sería:

3 5 1 9

En este ejercicio se pueden mezclar entradas y salidas con cómputos.

Finalidad: Trabajar con bucles que recorren los dígitos de un número. Dificultad Media.

31. El algoritmo de la multiplicación rusa es una forma distinta de calcular la multiplicación de dos números enteros $n * m$. Para ello este algoritmo va calculando el doble del multiplicador m y la mitad (sin decimales) del multiplicando n hasta que n tome el valor 1 y suma todos aquellos multiplicadores cuyos multiplicandos sean impares. Por ejemplo, para multiplicar 37 y 12 se harían las siguientes iteraciones

Iteración	Multiplicando	Multiplicador
1	37	12
2	18	24
3	9	48
4	4	96
5	2	192
6	1	384

Con lo que el resultado de multiplicar 37 y 12 sería la suma de los multiplicadores correspondientes a los multiplicandos impares (en negrita), es decir $37*12=12+48+384=444$

Cree un programa para leer dos enteros n y m y calcule su producto utilizando este algoritmo. No puede utilizarse en ningún momento el operador producto $*$.

Dificultad Media.

32. Amplíe el ejercicio 8 (año bisiesto). El programa pedirá los valores de dos años obligando a que estén entre el año cero y 2100. A continuación, el programa mostrará todos los años bisiestos comprendidos entre los años anteriores.

Finalidad: Practicar con filtros y ciclos básicos. Practicar con algoritmos más elaborados y eficientes. Reutilizar código ya escrito y verificado. Dificultad Media.

33. El método RLE (Run Length Encoding) codifica una secuencia de datos formada por series de valores idénticos consecutivos como una secuencia de parejas de números (valor de la secuencia y número de veces que se repite). Esta codificación es un mecanismo de compresión de datos (zip) sin pérdidas. Se aplica, por ejemplo, para comprimir los ficheros de imágenes en las que hay zonas con los mismos datos (fondo blanco, por ejemplo). Realice un programa que lea una secuencia de números naturales terminada con un número negativo y la codifique mediante el método RLE.

Entrada:	1 1 1 2 2 2 2 3 3 3 3 3 5 -1
	(tres veces 1, cinco veces 2, seis veces 3, una vez 5)
Salida:	3 1 5 2 6 3 1 5

Finalidad: Controlar en una iteración lo que ha pasado en la anterior. Dificultad Media.

34. Realice un programa que lea dos secuencias de enteros desde teclado y nos diga si todos los valores de la primera secuencia son mayores que todos los valores de la segunda secuencia.

Realice la lectura de los enteros dentro de sendos bucles sobre una única variable llamada dato. El final de cada secuencia viene marcado cuando se lee el 0.

Finalidad: Ejercitarse el uso de bucles. Dificultad Baja.

35. Se pide diseñar un programa para jugar a adivinar un número entre 1 y 100. El juego tiene que dar pistas de si el número introducido por el jugador está por encima o por debajo del número introducido. Como reglas de parada se consideran los siguientes dos casos:

- a) se ha acertado b) se decide abandonar el juego (decida cómo quiere especificar esta opción)

Para poder generar números aleatorios en un rango determinado será necesario incluir las siguientes instrucciones:

```
#include <iostream>
#include <ctime>
#include <cstdlib>
using namespace std;

int main(){
    const int MIN = 1,MAX = 100;
    const NUM_VALORES = MAX-MIN + 1;           // rango
```

```
int incognita;                                // número generado
time_t tiempo;

// Inicialización de la secuencia:
srand(time(&tiempo));

// Generación de un número aleatorio incognita:
// MIN <= incognita <= MAX
incognita = (rand() \% NUM_VALORES) + MIN;
```

La sentencia `srand(time(&tiempo))` debe ejecutarse una única vez al principio del programa y sirve para inicializar la secuencia de números aleatorios. Posteriormente, cada vez que se ejecute la sentencia `incognita = (rand() \% NUM_VALORES) + MIN;` se obtendrá un valor aleatorio (pseudoaleatorio).

Realizar el mismo ejercicio pero permitiendo jugar tantas veces como lo desee el jugador.

Dificultad Media.

36. Una empresa que tiene tres sucursales decide llevar la contabilidad de las ventas de sus productos a lo largo de una semana. Para ello registra cada venta con tres números, el identificador de la sucursal (1, 2 o 3), el código del producto codificado como un carácter (a, b ó c) y el número de unidades vendidas. Diseñar un programa que lea desde el teclado una serie de registros compuestos por **sucursal**, **producto**, **unidades** y diga cuál es la sucursal que más productos ha vendido. La serie de datos termina cuando la sucursal introducida vale -1. Por ejemplo, con la serie de datos

```
2 a 20
1 b 10
1 b 4
3 c 40
1 a 1
2 b 15
1 a 1
1 c 2
2 b 6
-1
```

Se puede ver que la sucursal que más productos ha vendido es la número 2 con 41 unidades totales. Para comprobar que el programa funciona correctamente, cread un fichero de texto y re-dirigid la entrada a dicho fichero.

Finalidad: Ver un bucle en el que se leen varios datos en cada iteración, pero sólo uno de ellos se usa como terminador de la entrada. Dificultad Media.

37. Un número entero n se dice que es *desgarrable* (torn) si al dividirlo en dos partes cualesquiera izda y dcha, el cuadrado de la suma de ambas partes es igual a n . Por ejemplo, 88209 es desgarrable ya que $(88 + 209)^2 = 88209$; 81 también lo es ya que $81 = (8 + 1)^2$. Cree un programa que lea un entero n e indique si es o no desgarrable.

Finalidad: Ejercitarse con los bucles. Dificultad Baja.

38. Un número entero de n dígitos se dice que es **narcisista** si se puede obtener como la suma de las potencias n -ésimas de cada uno de sus dígitos. Por ejemplo 153 y 8208 son números narcisistas porque $153 = 1^3 + 5^3 + 3^3$ (153 tiene 3 dígitos) y $8208 = 8^4 + 2^4 + 0^4 + 8^4$ (8208 tiene 4 dígitos). Construya un programa que, dado un número entero positivo, nos indique si el número es o no narcisista.

Finalidad: Ejercitarse con los bucles. Dificultad Media.

39. Todo lo que se puede hacer con un bucle `while` se puede hacer con un `do while`. Lo mismo ocurre al revés. Sin embargo, cada bucle se usa de forma natural en ciertas situaciones. El no hacerlo, nos obligará a escribir más código y éste será más difícil de entender. Para comprobarlo, haced lo siguiente:

- Modifique la solución del ejercicio 19 de forma que el filtro de entrada usado para leer la variable `tope`, se haga con un bucle pre-test `while`.
- Modifique la solución del ejercicio 20 sustituyendo el bucle `while` por un `do while`. Observad que debemos considerar el caso en el que el número de años leído fuese cero.

Finalidad: Enfatizar la necesidad de saber elegir entre un bucle pre-test o un bucle post-test. Dificultad Media.

40. Calcule mediante un programa en C++ la función potencia x^n , y la función factorial $n!$ con n un valor entero y x un valor real. No pueden usarse las funciones de la biblioteca `cmath`, por lo que tendrá que implementar los cálculos con los bucles necesarios.

El factorial de un entero n se define de la forma siguiente:

$$0! = 1$$

$$n! = 1 \times 2 \times 3 \times \cdots \times n, \quad \forall n \geq 1$$

Escriba un programa de prueba que lea un número entero n obligando a que esté en el intervalo $[1, 20]$. A continuación lea un valor real x y calcule e imprima en pantalla el factorial de n y la potencia de x elevado a n .

Finalidad: Trabajar con bucles controlados por contador. Dificultad Baja.

41. Calcule mediante un programa en C++ el combinatorio $\binom{n}{m}$ con n, m valores enteros. No pueden usarse las funciones de la biblioteca `cmath`.

RELACIÓN DE PROBLEMAS II. Estructuras de Control

El combinatorio de n sobre m (con $n \geq m$) es un número entero que se define como sigue:

$$\binom{n}{m} = \frac{n!}{m! (n - m)!}$$

Finalidad: Trabajar con bucles controlados por contador. Dificultad Media.

42. Escriba un programa que lea cuatro valores de tipo char (`min_izda`, `max_izda`, `min_dcha`, `max_dcha`) e imprima las parejas que pueden formarse con un elemento del conjunto `{min_izda ... max_izda}` y otro elemento del conjunto `{min_dcha ... max_dcha}`. Por ejemplo, si `min_izda = b`, `max_izda = d`, `min_dcha = j`, `max_dcha = m`, el programa debe imprimir las parejas que pueden formarse con un elemento de `{b c d}` y otro elemento de `{j k l m}`, es decir:

bj bk bl bm
cj ck cl cm
dj dk dl dm

Finalidad: Ejercitarse los bucles anidados. Dificultad Baja.

43. Cree un programa que ofrezca en pantalla la siguiente salida:

1 2 3 4 5 6
2 3 4 5 6
3 4 5 6
4 5 6
5 6
6

Finalidad: Ejercitarse los bucles anidados. Dificultad Baja.

44. Cree un programa que ofrezca en pantalla la siguiente salida:

1 2 3 4 5 6
2 3 4 5 6 7
3 4 5 6 7 8
4 5 6 7 8 9
5 6 7 8 9 10
6 7 8 9 10 11

Finalidad: Ejercitarse los bucles anidados. Dificultad Media.

45. Modifique los dos ejercicios anteriores para que se lea desde teclado el valor inicial y el número de filas a imprimir. En los ejemplos anteriores, el valor inicial era 1 y se imprimían un total de 6 filas.

Finalidad: Ejercitarse los bucles anidados. Dificultad Media.

46. Construya un programa que lea un valor T y calcule la siguiente sumatoria:

$$\sum_{i=1}^{i=T} i! = \sum_{i=1}^{i=T} \left(\prod_{j=1}^{j=i} j \right)$$

Por ejemplo, para $T = 4$, la operación a realizar es:

$$1! + 2! + 3! + 4!$$

es decir:

$$1 + (1 * 2) + (1 * 2 * 3) + (1 * 2 * 3 * 4)$$

Ejemplo de entrada: 3 — Salida correcta: 9

Ejemplo de entrada: 4 — Salida correcta: 33

Ejemplo de entrada: 6 — Salida correcta: 873

Finalidad: Ejercitarse con los bucles anidados. Dificultad Media.

47. Resuelva el ejercicio 46 sin utilizar bucles anidados, es decir, debe usar un único bucle.

Finalidad: Aprovechar en una iteración los cálculos hechos en la iteración anterior. Dificultad Media.

48. Recupere la solución del ejercicio 10 (función gaussiana) de la relación de problemas I. Se pide crear un menú principal para que el usuario pueda elegir las siguientes opciones:

**Introducir parámetros de la función (esperanza y desviación)
Salir del programa**

Si el usuario elige la opción de salir, el programa terminará; si elige la opción de introducir los parámetros, el programa leerá los dos parámetros (esperanza y desviación). La media puede ser un valor cualquiera, pero la desviación ha de ser un número positivo. A continuación, el programa presentará un menú con las siguientes opciones:

**Introducir rango de valores de abscisas
Volver al menú anterior (el menú principal)**

Si el usuario elige volver al menú anterior, el programa debe presentar el primer menú (el de la introducción de los parámetros). Si el usuario elige introducir los valores de abscisas, el programa le pedirá un valor **minimo**, un valor **maximo** (ha de ser mayor que **minimo**) y un **incremento** y mostrará el valor de la función gaussiana en todos los valores de x (la abscisa) entre **minimo** y **maximo** a saltos de **incremento**, es decir, **minimo**, **minimo + incremento**, **minimo + 2*incremento**, ..., hasta llegar, como mucho, a **maximo**. Después de mostrar los valores de la función, el programa volverá al menú de introducción del rango de valores de abscisas.

Ejemplo de entrada: P 12 5 R 11 13 0.5 V S (Se han elegido las letras P para introducir parámetros, R para introducir el rango, V para volver del menú secundario al principal y S para salir del programa)

-- Salida correcta:

$$\begin{aligned}f(11) &= 0.0782085 \\f(11.5) &= 0.0793905 \\f(12) &= 0.0797885 \\f(12.5) &= 0.0793905 \\f(13) &= 0.0782085\end{aligned}$$

Finalidad: Ejercitarse los bucles anidados. Dificultad Media.

49. Diremos que un número entero positivo es secuenciable si se puede generar como suma de números consecutivos (al menos dos). Por ejemplo, $6 = 1+2+3$, $15 = 7+8$. Esta descomposición no tiene por qué ser única. Por ejemplo, $15 = 7+8 = 4+5+6 = 1 + 2 + 3 + 4 + 5$. Escriba un programa que lea un entero $n \geq 1$ e imprima todas las descomposiciones posibles. En este ejercicio puede mezclar operaciones de E/S y C dentro del mismo bucle.

Como curiosidad, los únicos números con 0 descomposiciones son las potencias de 2.

Ejemplo de entrada: 6 -- Salida correcta: 1 2 3

Ejemplo de entrada: 15 -- Salida correcta: 7 8 / 4 5 6 / 1 2 3 4 5

Finalidad: Ejercitarse los bucles anidados. Dificultad Media.

50. (*Examen Septiembre 2014*) ¿Cuántas veces aparece el dígito 9 en todos los números que hay entre el 1 y el 100? Por ejemplo, el 9 aparece una vez en los números 19 y 92 mientras que aparece dos veces en el 99. Pretendemos diseñar un algoritmo que responda a esta sencilla pregunta, pero de forma suficientemente generalizada. Para ello, se pide construir un programa que lea una **cifra** (entre 1 y 9), dos enteros **min** y **max** y calcule el número de apariciones del dígito **cifra** en los números contenidos en el intervalo cerrado **[min, max]**.

Finalidad: Ejercitarse los bucles anidados. Dificultad Baja.

51. Supongamos una serie numérica cuyo término general es:

$$a_i = a_1 r^{i-1}$$

Es decir, la serie la forman los siguientes términos:

$$\begin{aligned}a_1 &= a_1 \\a_2 &= a_1 r \\a_3 &= a_1 r^2\end{aligned}$$

$$a_4 = a_1 r^3$$

...

Se pide crear un programa que lea desde teclado r , el primer elemento a_1 y el tope k y calcule la suma de los primeros k valores de la serie, es decir:

$$\sum_{i=1}^{i=k} a_i$$

Se proponen dos alternativas:

- a) Realice la suma de la serie usando la función `pow` para el cómputo de cada término a_i . Los argumentos de `pow` no pueden ser ambos enteros, por lo que forzaremos a que la base (por ejemplo) sea `double`, multiplicando por `1.0`.
- b) Si analizamos la expresión algebraica de la serie numérica, nos damos cuenta que es una *progresión geométrica* ya que cada término de la serie queda definido por la siguiente expresión:

$$a_{i+1} = a_i * r$$

Es decir, una progresión geométrica es una secuencia de elementos en la que cada uno de ellos se obtiene multiplicando el anterior por una constante denominada razón o factor de la progresión.

Cree el programa pedido usando esta fórmula. NO puede utilizarse la función `pow`.

¿Qué solución es preferible en términos de eficiencia?

Finalidad: Trabajar con bucles que aprovechan cómputos realizados en la iteración anterior. Dificultad Baja.

52. Reescribid la solución a los ejercicios 19 (divisores) y 20 (interés) usando un bucle `for`

Finalidad: Familiarizarnos con la sintaxis de los bucles for. Dificultad Baja.

53. Diseñar un programa para calcular la suma de los 100 primeros términos de la sucesión siguiente:

$$a_i = \frac{(-1)^i(i^2 - 1)}{2i}$$

No puede usarse la función `pow`. Hacedlo calculando explícitamente, en cada iteración, el valor $(-1)^i$ (usad un bucle `for`). Posteriormente, resolvedlo calculando dicho valor a partir del calculado en la iteración anterior, es decir, $(-1)^{i-1}$.

Finalidad: Enfatizar la conveniencia de aprovechar cómputos realizados en la iteración anterior. Dificultad Media.

54. Sobre la solución del ejercicio 20 de esta relación de problemas, se pide lo siguiente. Supondremos que sólo pueden introducirse intereses enteros (1, 2, 3, etc). Se pide calcular el capital obtenido al término de cada año, pero realizando los cálculos para todos los tipos de interés enteros menores o iguales que el introducido (en pasos de 1). Por ejemplo, si el usuario introduce un interés igual a 5 y un número de años igual a 3, hay que mostrar el capital ganado al término de cada uno de los tres años a un interés del 1 %, a continuación, lo mismo para un interés del 2 % y así sucesivamente hasta llegar al 5 %. El programa debe mostrar una salida del tipo:

Cálculos realizados al 1%:

Dinero obtenido en el año número 1 = 2020
Dinero obtenido en el año número 2 = 2040.2
Dinero obtenido en el año número 3 = 2060.6

Cálculos realizados al 2%:

Dinero obtenido en el año número 1 = 2040
Dinero obtenido en el año número 2 = 2080.8
Dinero obtenido en el año número 3 = 2122.42
.....

Finalidad: Empezar a trabajar con bucles anidados. Dificultad Baja.

55. Implemente un programa que sea capaz de “dibujar” rectángulos utilizando un símbolo (un carácter) dado. El usuario ingresará el símbolo *simb*, la altura *M* y el ancho *N* del rectángulo. Por ejemplo, siendo *simb* =*, *M* = 3 y *N* = 5, el dibujo tendría la siguiente forma:

Finalidad: Ejercitarse con los bucles anidados. Dificultad Baja.

56. Implemente un programa que sea capaz de “dibujar” pinos utilizando asteriscos “*”. El usuario ingresará el ancho de la base del pino (podemos asumir que es un número impar). Supongamos que se ingresa 7, entonces el dibujo tendrá la siguiente forma:

*

Finalidad: Ejercitarse con los bucles anidados. Dificultad Media.

57. Se dice que un número natural es feliz si cumple que si sumamos los cuadrados de sus dígitos y seguimos el proceso con los resultados obtenidos, finalmente obtenemos uno (1) como resultado. Por ejemplo, el número 203 es un número feliz ya que $2^2 + 0^2 + 3^2 = 13 \rightarrow 1^2 + 3^2 = 10 \rightarrow 1^2 + 0^2 = 1$.

Se dice que un número es feliz de grado k si se ha podido demostrar que es feliz en un máximo de k iteraciones. Se entiende que una iteración se produce cada vez que se elevan al cuadrado los dígitos del valor actual y se suman. En el ejemplo anterior, 203 es un número feliz de grado 3 (además, es feliz de cualquier grado mayor o igual que 3)

Escribir un programa que diga si un número natural n es feliz para un grado k dado de antemano. Tanto n como k son valores introducidos por el usuario.

Finalidad: Ejercitarse con los bucles anidados. Dificultad Media.

58. Realizar un programa para calcular los valores de la función:

$$f(x) = \sqrt{\frac{3x + x^2}{1 - x^2}}$$

para valores de x enteros en el rango [-3..3].

Dificultad Baja.

59. Realizar un programa para calcular los valores de la función:

$$f(x, y) = \frac{\sqrt{x}}{y^2 - 1}$$

para los valores de (x, y) con $x = -50, -48, \dots, 48, 50$ y $y = -40, -39, \dots, 39, 40$, es decir queremos mostrar en pantalla los valores de la función en los puntos

$$(-50, 40), (-50, -39), \dots, (-50, 40), (-48, 40), (-48, -39), \dots, (50, 40)$$

Dificultad Baja.

60. Diseñar un programa que presente una tabla de grados C a grados Fahrenheit ($F=9/5C+32$) desde los 0 grados a los 300, con incremento de 20 en 20 grados.

Dificultad Baja.

61. Diseñar un programa que lea caracteres desde la entrada y los muestre en pantalla, hasta que se pulsa el '.' y diga cuántos separadores se han leído (espacios en blanco ' ', tabuladores '\t' y caracteres de nueva línea '\n').

Dificultad Baja.

62. Realizar un programa para calcular la suma de los términos de la serie

$$1 - 1/2 + 1/4 - 1/6 + 1/8 - 1/10 + \dots - 1/(2n-1) + 1/(2n)$$

para un valor n dado.

Dificultad Baja.

63. Se decide informatizar el acta de un partido de baloncesto para saber qué equipo es el ganador del partido. El acta contiene una serie de anotaciones formadas por una pareja de números cada una, con el dorsal del jugador y el número de puntos conseguidos teniendo en cuenta que la última anotación es un valor -1. Por ejemplo

1 2 4 1 4 1 2 3 6 2 3 2 5 2 5 1 1 3 -1

El programa deberá indicar si ha ganado el equipo 1 (con los dorsales 1, 2 y 3) o el equipo 2 (dorsales 4, 5 y 6) o han empatado.

Por ejemplo, con la entrada anterior, gana el equipo 1.

Dificultad Baja.

64. La Unión Europea ha decidido premiar al país que más toneladas de hortalizas exporte a lo largo del año. Se dispone de un registro de transacciones comerciales en el que aparecen tres valores en cada apunte. El primer valor es el indicativo del país (E: España, F: Francia y A: Alemania), el segundo valor es un indicativo de la hortaliza que se ha vendido en una transacción (T: Tomate, P: Patata, E: Espinaca) y el tercer valor indica las toneladas que se han vendido en esa transacción. Diseñar un programa que lea desde el teclado este registro, el cual termina siempre al leer un país con indicativo '@', y que diga qué país es el que más hortalizas exporta y las toneladas que exporta.

Por ejemplo, con la entrada

E T 10 E T 4 E P 1 E P 1 E E 2 F T 15 F T 6 F P 20 A E 40 @

el país que más vende es Francia con un total de 41 toneladas.

Dificultad Baja.

65. Se pide leer dos enteros sabiendo que el primero no tiene un tamaño fijo y que el segundo siempre es un entero de dos dígitos. Se pide comprobar si el segundo está contenido en el primero. Entendemos que está contenido si los dos dígitos del segundo entero están en el primer entero de forma consecutiva y en el mismo orden. Por ejemplo, 89 está contenido en 7890, en 7789 y en 8977 pero no en 7980.

Dificultad Media.

66. Se dice que un número es triangular si se puede poner como la suma de los primeros m valores enteros, para algún valor de m . Por ejemplo, 6 es triangular ya que $6 = 1 + 2 + 3$. Se pide construir un programa que obtenga todos los números triangulares que hay menores que un entero **tope** introducido desde teclado.

Dificultad Baja.

67. Escriba un programa que lea por teclado un número entero positivo **tope** y muestre por pantalla el factorial de los **tope** primeros números enteros. Recuerda que el factorial de un número entero positivo n es igual al producto de los enteros positivos del 1 al n .

Dificultad Baja.

68. Construya un programa para comprobar si las letras de una palabra se encuentran dentro de otro conjunto de palabras. Los datos se leen desde un fichero de la forma siguiente: el fichero contiene, en primer lugar un total de 3 letras que forman la palabra a buscar, por ejemplo **f e o**. Siempre habrá, exactamente, tres letras. A continuación, el fichero contiene el conjunto de palabras en el que vamos a buscar. El final de cada palabra viene determinado por la aparición del carácter '**@**', y el final del fichero por el carácter '**#**'. La búsqueda tendrá las siguientes restricciones:

- Deben encontrarse las tres letras
- Debe respetarse el orden de aparición. Es decir, si por ejemplo encontramos la '**f**' en la segunda palabra, la siguiente letra a buscar '**e**' debe estar en una palabra posterior a la segunda.
- Una vez encontremos una letra en una palabra, ya no buscaremos más letras en dicha palabra.
- No nos planteamos una búsqueda barajando todas las posibilidades, en el sentido de que una vez encontrada una letra, no volveremos a buscarla de nuevo.

Entrada:	<pre>f e o h o l a @ m o f e t a @ <- f c o f i a @ c e r r o @ <- e p e r a @ c o s a @ <- o h o y @ #</pre>
----------	---

En este caso, sí se encuentra.

Dificultad Media.

69. Un número perfecto es aquel que es igual a la suma de todos sus divisores positivos excepto él mismo. El primer número perfecto es el 6 ya que sus divisores son 1, 2 y 3 y $6=1+2+3$. Escribir un programa que muestre el mayor número perfecto que sea menor a un número dado por el usuario.

Dificultad Media.

70. Escribir un programa que encuentre dos enteros n y m mayores que 1 que verifiquen lo siguiente:

$$\sum_{i=1}^m i^2 = n^2$$

Dificultad Media.

71. En matemáticas, la **sucesión de Fibonacci** (a veces mal llamada *serie* de Fibonacci) es la siguiente sucesión infinita de números naturales:

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \dots$$

La sucesión comienza con los números 1 y 1, y a partir de éstos, cada término puede calcularse como la suma de los dos anteriores. A los elementos de esta sucesión se les llama *números de Fibonacci*.

El número de Fibonacci de orden n , al que llamaremos f_n se puede definir mediante la siguiente relación de recurrencia:

- $f_n = f_{n-1} + f_{n-2}$ para $n > 2$
- $f_1 = f_2 = 1$

Esta sucesión fue descrita en Europa por Leonardo de Pisa, matemático italiano del siglo XIII también conocido como Fibonacci. Tiene numerosas aplicaciones en ciencias de la computación, matemáticas y teoría de juegos. También aparece en diversas configuraciones biológicas.

Escribir un programa que calcule el número de Fibonacci de orden n , donde n es un valor introducido por el usuario. A continuación, el programa solicitará un nuevo valor, k , y mostrará todos los números de Fibonacci $f_1, f_2, f_3, \dots, f_k$.

Finalidad: Trabajar con bucles controlados por contador. Dificultad Media.

72. El número áureo se conoce desde la Antigüedad griega y aparece en muchos temas de la geometría clásica. La forma más sencilla de definirlo es como el único número positivo ϕ que cumple que $\phi^2 - \phi = 1$ y por consiguiente su valor es $\phi = \frac{1 + \sqrt{5}}{2}$.

Se pueden construir aproximaciones al número áureo mediante la fórmula $a_n = \frac{f_{n+1}}{f_n}$ siendo f_n el número de Fibonacci de orden n (ver problema 71).

La sucesión de valores así calculada proporciona, alternativamente, valores superiores e inferiores a ϕ , siendo cada vez más cercanos a éste, y por lo tanto la diferencia entre a_n y ϕ es cada vez más pequeña conforme n se hace mayor.

Escribir un programa que calcule el menor valor de n que hace que la aproximación dada por a_n difiera en menos de δ del número ϕ , sabiendo que $n \geq 1$.

La entrada del programa será el valor de δ y la salida el valor de n . Por ejemplo, para un valor de $\delta = 0,1$ el valor de salida es $n = 4$

Dificultad Media.

73. Una *sucesión alícuota* es una sucesión iterativa en la que cada término es la suma de los divisores propios del término anterior. La sucesión alícuota que comienza con el entero positivo k puede ser definida formalmente mediante la función divisor σ_1 de la siguiente manera:

$$\begin{aligned}s_0 &= k \\ s_n &= \sigma_1(s_{n-1}) - s_{n-1}\end{aligned}$$

Por ejemplo, la sucesión alícuota de 10 es 10, 8, 7, 1, 0 porque:

$$\begin{aligned}\sigma_1(10) - 10 &= 5 + 2 + 1 = 8 \\ \sigma_1(8) - 8 &= 4 + 2 + 1 = 7 \\ \sigma_1(7) - 7 &= 1 \\ \sigma_1(1) - 1 &= 0\end{aligned}$$

Aunque muchas sucesiones alícuotas terminan en cero, otras pueden no terminar y producir una sucesión alícuota periódica de período 1, 2 o más. Está demostrado que si en una sucesión alícuota aparece un *número perfecto* (como el 6) se produce una sucesión infinita de período 1. Un *número amigable* produce una sucesión infinita de período 2 (como el 220 ó 284).

Escribir un programa que lea un número natural menor que 1000 y muestre su sucesión alícuota. Hay que tener en cuenta que en ocasiones se pueden producir sucesiones infinitas, por lo que en estos casos habrá que detectarlas e imprimir puntos suspensivos cuando el período se repita. Solo hay que considerar períodos infinitos de longitud 2 como máximo. Por ejemplo; para el número 6, se imprimiría: 6, 6, ...; y para el número 220, se imprimiría: 220, 284, 220, 284,

Finalidad: Practicar los bucles anidados y controlar las condiciones de parada a partir de lo sucedido en iteraciones pasadas.. Dificultad Media.

RELACIÓN DE PROBLEMAS III. Funciones y Clases

Problemas sobre funciones

1. Encuentre los errores de las siguientes funciones:

```
int ValorAbsoluto (int entero) {    void Imprime(double valor) {  
    if (entero < 0)                      double valor;  
        entero = -entero;  
    else                                cout << valor;  
        return entero;                  }  
    }  
  
void Cuadrado (int entero) {          bool EsPositivo(int valor) {  
    return entero*entero;                if (valor > 0)  
    }                                    return true;  
}                                     }
```

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

2. Reescriba la solución del ejercicio **40** (factorial y potencia) de la Relación de Problemas II, modularizándola con funciones.

Para el factorial, use la función **Factorial** vista en las transparencias de clase. Para el cómputo de la potencia, defina la función **Potencia**.

Finalidad: Familiarizarnos con la definición de funciones y el paso de parámetros. Dificultad Baja.

3. En las transparencias de clase se ha visto la función **Max3** que calculaba el máximo de tres valores enteros.

Defina ahora la función **Max** que calcule el máximo de dos valores enteros y cambie la implementación de **Max3** para que llame a la función **Max**.

Construya un programa principal que llame a **Max3** con tres valores leídos desde teclado.

Finalidad: Familiarizarnos con las llamadas entre funciones. Dificultad Baja.

4. En el ejercicio **24** de la Relación de Problemas II, se pedía leer un número en un rango. Defina una función **LeeIntRango** para este propósito. Para ello, dicha función debe ir leyendo números enteros (de tipo **int**) desde la entrada por defecto, hasta que se

lea un valor correcto que pertenezca al rango `[min, max]` (no hay ningún límite en el número de intentos). La función devolverá dicho valor.

Escriba un pequeño programa de prueba que lea dos números cualesquiera `min` y `max`. Supondremos que el valor introducido de `max` es correcto, es decir, que será mayor o igual que `min`. A continuación lea 3 valores en dicho rango y finalmente calcule la suma de dichos valores.

Ejemplo de entrada: 3 6 -1 2 3 7 2 1 4 4 -- Salida correcta: 11

Ejemplo de entrada: 3 6 3 4 4 -- Salida correcta: 11

*Finalidad: Familiarizarnos con la definición de funciones y el paso de parámetros.
Dificultad Baja.*

5. Recupere la solución del ejercicio 4. Defina ahora una función `LeeIntMayorIgualQue` para leer un entero mayor o igual que un número dado (éste será un parámetro a la función). Para ello, dicha función debe ir leyendo números enteros (de tipo `int`) desde la entrada por defecto, hasta que se lea un valor correcto que sea mayor o igual que el número especificado. La función devolverá dicho valor.

Utilice esta función para leer el valor de `max` del ejercicio 4, obligando a que sea mayor que `min`.

Ejemplo de entrada: 3 2 1 6 -1 2 3 7 2 1 4 4

-- Salida correcta: 11

Ejemplo de entrada: 3 6 3 4 4

-- Salida correcta: 11

*Finalidad: Familiarizarnos con la definición de funciones y el paso de parámetros.
Dificultad Baja.*

6. Reescriba la solución del ejercicio 46 que calcula la suma de los primeros T factoriales. Para ello, debe leer el valor T usando la función `LeeIntRango` del ejercicio 4 para obligar a que esté en el intervalo $[1, 20]$.

Debe definir la función `SumaFactoriales` que calcule la suma pedida. Implemente dos versiones de esta función:

- En una primera versión, la función `SumaFactoriales` debe llamar a la función `Factorial`, para realizar la suma tal y como se indica en el ejercicio 46
- En una segunda versión, la función `SumaFactoriales` debe realizar la suma de forma directa tal y como se indica en el ejercicio 47. Ponga dentro de un comentario la primera versión.

Finalidad: Familiarizarnos con la llamada entre funciones. Dificultad Baja.

7. Retome la solución del ejercicio 48 (Gaussiana) y modifíquela introduciendo funciones dónde crea conveniente. Al menos debe definir la función **gaussiana** para que calcule el valor de la ordenada, para unos valores concretos de abscisa, esperanza y desviación.

*Finalidad: Familiarizarnos con la definición de funciones y el paso de parámetros.
Dificultad Baja.*

8. Retome la solución del ejercicio 7 (Gaussiana)

Ahora estamos interesados en obtener el área que cubre la función gaussiana 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:

$$CDF(x) = \int_{-\infty}^x \text{gaussiana}(t)dt$$

Puede probar algunos valores ejecutando la siguiente calculadora online:

<https://www.easycalculation.com/statistics/normal-distribution.php>

El valor de x hay que introducirlo en el apartado *Below*.

Para no tener que implementar el concepto de integral, vamos a recurrir a una aproximación numérica para obtener **CDF(x)**. Puede consultarse en la Wikipedia (buscar *Normal distribution*) que la siguiente fórmula proporciona una aproximación al valor de **CDF(x)**:

$$CDF(x) = \text{Área hasta } (x) \approx 1 - \text{gaussiana}(x)(b_1 t + b_2 t^2 + b_3 t^3 + b_4 t^4 + b_5 t^5)$$

dónde:

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

$$b_3 = 1,781477937 \quad b_4 = -1,821255978 \quad b_5 = 1,330274429$$

Cree otra función para calcular el área hasta un punto cualquiera x , es decir, **CDF(x)**, usando la anterior aproximación. Para implementar esta función, use la función **Potencia** del ejercicio 2 cuando tenga que calcular los términos t^i .

Modifique el programa principal del ejercicio 7 para que llame a la función **CDF(x)** e imprima las ordenadas correspondientes a las abscisas **minimo**, **minimo + incremento**, **minimo + 2*incremento**, etc.

Ejemplo de entrada: P 12 5 R 11 13 0.5 V S

-- Salida correcta:

$$f(11)=0.0782085$$

$$CDF(11)=0.998414$$

$$f(11.5)=0.0793905$$

$$CDF(11.5)=0.998573$$

$$f(12)=0.0797885$$

$$CDF(12)=0.998727$$

$$f(12.5)=0.0793905$$

$$CDF(12.5)=0.998875$$

$$f(13)=0.0782085$$

$$CDF(13)=0.999016$$

Finalidad: Entender las llamadas entre funciones y la importancia de la ocultación de información. Dificultad Baja.

9. Retome la solución del ejercicio 10 (parking) de la Relación de Problemas II. Se quiere extender para poder trabajar con varios parkings o con varias tarifas distintas. Supondremos que en todos los casos, se tiene el mismo número de tramos (4) aunque puede variar la cuantía a tarifar por minutos y los límites de cada uno de los tramos.

Para ello, se pide definir la función Tarifa que obtenga la tarifa final aplicable a cualquier caso. En concreto, en este ejercicio, se van a leer sólo dos casos, correspondientes a dos parkings o tarificaciones distintas. Se pide por tanto construir un programa que lea los siguientes datos:

- En primer lugar el programa lee los datos de cada uno de los dos casos, es decir, los límites de los tramos y las tarifas que se aplican en cada tramo (ver tabla debajo)
- A continuación, se leen varios pares de instantes de entrada y salida. Se leen en el orden instante de entrada (hora, minuto y segundo) e instante de salida.
La entrada de datos finaliza cuando se introduce un -1 como hora de entrada.

El programa imprimirá la tarifa resultante de cada uno de los parkings para cada par de instantes de entrada y salida, así como la suma total recaudada en cada caso.

Por ejemplo:

```

30      -> Límite 1 del parking 1
90      -> Límite 2 del parking 1
120     -> Límite 3 del parking 1
660     -> Límite 4 del parking 1
0.0412 -> Tarifa Tramo 1 del parking 1
0.0370 -> Tarifa Tramo 2 del parking 1
0.0311 -> Tarifa Tramo 3 del parking 1
0.0305 -> Tarifa Tramo 4 del parking 1
31.55   -> Tarifa día completo del parking 1
35      -> Límite 1 del parking 2

```

RELACIÓN DE PROBLEMAS III. Funciones y Clases

```
85      -> Limite 2 del parking 2
110     -> Limite 3 del parking 2
660     -> Limite 4 del parking 2
0.0402  -> Tarifa Tramo 1 del parking 2
0.0375  -> Tarifa Tramo 2 del parking 2
0.0319  -> Tarifa Tramo 3 del parking 2
0.0315  -> Tarifa Tramo 4 del parking 2
32      -> Tarifa día completo del parking 2
2 1 30  -> Entra a las 2 de la madrugada, 1 minuto, 30 segundos
4 2 50  -> Sale a las 4 de la madrugada, 2 minutos y 50 segundos
2 2 5   -> Entra a las 2 de la madrugada, 2 minutos, 5 segundos
4 3 7   -> Sale a las 4 de la madrugada, 3 minutos, 7 segundos
-1      -> Fin de la entrada de datos.
```

Ejemplo de entrada:

```
30 90 120 660 0.0412 0.0370 0.0311 0.0305 31.55
35 85 110 660 0.0402 0.0375 0.0319 0.0315 32
2 1 30 4 2 50
2 1 30 3 41 31
2 1 30 5 41 31
2 1 30 23 1 1  -1
```

-- Salida correcta:

```
4.4195 4.4262
3.767 3.7605
7.439 7.5445
31.55 32
```

```
47.1755
47.731
```

Finalidad: Diseño de una función. Dificultad Media.

10. Retome la solución del ejercicio 23 (población) de la Relación de Problemas II. Rescríbalo usando las funciones LeeIntRango del ejercicio 4 para leer los valores de las tasas y LeeIntMayorIgualQue del ejercicio 5 para leer el número de años que sea positivo. Defina también sendas funciones para calcular los dos valores que se piden en el ejercicio, a saber, el número de habitantes después de tres años y el número de años que pasarán hasta doblar la población inicial. Intente diseñar las funciones para que sean lo más generales posible.

Ejemplo de entrada:

```
1375570814 2000 32 2000 2000 12 7 -4 -4 3
```

RELACIÓN DE PROBLEMAS III. Funciones y Clases

-- Salida correcta: 1490027497 27 2824131580

Finalidad: Diseño de una función. Dificultad Baja.

11. Retome la solución de los ejercicios 11 y 28 (servicio atención telefónica) de la Relación de Problemas II. Recordemos que el criterio de subida salarial era el siguiente:

Entre 20 y 30 casos resueltos:	+3%
Más de 30 casos resueltos:	+4%
Grado de satisfacción >= 4:	+2%

Defina una función `SalarioFinal` que calcule el salario final del trabajador, en función de los datos anteriores.

Al igual que se pedía en el ejercicio 28 debe ir leyendo los datos de tres empleados en el siguiente orden:

```
7.5      <- Salario de 7.5 euros por hora (el mismo para todos)
2 124 1 3 <- Empleado 2, 124'', resuelto, grado sat: 3
1 32 0 0  <- Empleado 1, 32'', no resuelto, grado sat: 0
2 26 0 2  <- Empleado 2, 26'', no resuelto, grado sat: 2
-1        <- Fin de entrada de datos
```

El número de horas trabajadas de cada empleado será un número real y se calculará en función de la suma total de segundos dedicados a cada llamada telefónica (la compañía no paga por el tiempo de estancia en la empresa sino por el tiempo dedicado a resolver casos)

El programa debe llamar a la función `SalarioFinal` para calcular el salario final de cada uno de los tres empleados y los debe mostrar en pantalla.

Puede utilizar el fichero de datos `datos_atencion_telefonica.txt` disponible en `decsai`. La salida correcta para este fichero es
1016.196 118.287 128.893

*Finalidad: Familiarizarnos con la definición de funciones y el paso de parámetros.
Dificultad Media.*

12. Retome la solución del ejercicio 11 de esta Relación de Problemas. Modifíquela para tener en cuenta que los límites correspondientes a los casos resueltos (20 y 30) y el grado de satisfacción media (4), así como los porcentajes de incrementos correspondientes (3%, 4% y 2%) ya no son constantes sino que pueden variar.

Por lo tanto, debe leer desde teclado dichos valores límites (justo después del salario por hora y en el orden indicado anteriormente) y cambiar la función definida en el ejercicio 11 para que tenga en cuenta este cambio.

Puede utilizar el fichero de datos

`datos_atencion_telefonica_lmites_variables.txt`

disponible en `decsai`. La salida correcta para este fichero es
1016.196 118.287 128.893

*Finalidad: Familiarizarnos con la definición de funciones y el paso de parámetros.
Dificultad Baja.*

13. Implemente la solución del ejercicio 38 (Narcisista) de la relación de problemas II, usando funciones.

*Finalidad: Familiarizarnos con la definición de funciones y el paso de parámetros.
Dificultad Baja.*

14. Escriba una función en C++ `LeeOpcion2Alternativas` que imprima en pantalla un mensaje, lea una opción como un carácter y sólo permita aceptar los caracteres '`S`' o '`N`' (mayúscula o minúscula). ¿Qué debería devolver la función? ¿El carácter leído o un `bool`? Aplique esta función en la solución del ejercicio 17 (Renta bruta y neta) de la relación de problemas II, para leer si una persona es pensionista o si es autónomo.

*Finalidad: Familiarizarnos con la definición de funciones y el paso de parámetros.
Dificultad Baja.*

15. A un trabajador le pagan según sus horas trabajadas y la tarifa está a un valor por hora. Si la cantidad de horas trabajadas es mayor de 40 horas, la tarifa por hora se incrementa en un 50 % para las horas extras (las que haya por encima de 40). Construir una función que dado el número total de horas trabajadas y el precio por hora, devuelva el salario del trabajador.

Finalidad: Familiarizarnos con la definición de funciones y paso de parámetros. Dificultad Baja.

16. Cree las siguientes funciones relacionadas con la progresión geométrica que se vio en el ejercicio 51 de la relación de problemas II. Analice cuáles deben ser los parámetros a estas funciones.

- a) Una función `SumaHasta` que calcule la suma de los primeros k valores de una progresión geométrica.

Para implementarla, use el mismo algoritmo (con un bucle `for`) que se vio como solución del ejercicio 51 de la relación de problemas II.

- b) Una función `ProductoHasta` para que multiplique los k primeros elementos de la progresión, aplicando la siguiente fórmula:

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

Observe que no se pide calcular los productos acumulados en un bucle sino que simplemente evalúe la expresión $\sqrt{(a_1 a_k)^k}$ que le da directamente el producto de los k primeros términos.

- c) Una función **SumaHastaInfinito** para calcular la suma hasta infinito, según la siguiente fórmula:

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

De nuevo, observe que sólo hay que aplicar la expresión $\frac{a_1}{1 - r}$ para obtener la suma pedida. Esta fórmula sólo se puede aplicar cuando el valor absoluto de la razón es menor o igual que 1, ya que, en caso contrario, la suma saldría infinito.

Cree un programa principal que llame a estas funciones.

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

17. Amplie el ejercicio 16 cambiando la implementación de la función **SumaHasta**. Para ello, en vez de usar un bucle aplicamos la siguiente fórmula que nos da la sumatoria aplicando únicamente cinco operaciones:

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

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

Nota. Calculad la potencia (r^k) con la función **pow** y hacerlo también usando la función **Potencia** definida en el ejercicio 2 de esta Relación de Problemas.

Hay que destacar que el cómputo de la potencia 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 de la función **SumaHasta**, podemos cambiar su implementación sin tener que cambiar ni una línea de código del **main**.

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

18. Se pide construir las siguientes funciones:

- Una función que compruebe si un carácter es una mayúscula:

```
bool EsMayuscula(char caracter)
```

- Una función que realice un filtro de entrada para mayúsculas, es decir, dentro de la función se van leyendo caracteres (con `cin`) en un bucle hasta que se introduzca una mayúscula cualquiera o hasta que se introduzca un carácter terminador (asuma que dicho carácter es #)

La cabecera de la función será la siguiente:

```
char LeeMayuscula()
```

Esta función debe llamar a la anterior `EsMayuscula`. En el caso de que el carácter leído sea el terminador, la función devolverá ese mismo valor (#)

Construya ahora un programa principal que vaya leyendo caracteres, para lo cual debe llamar a la función `LeeMayuscula`. La entrada de datos terminará cuando se introduzca el terminador # y el programa debe mostrar en pantalla el número total de mayúsculas que se han introducido.

Puede suponer que no se introducen espacios en blanco.

Por ejemplo, si la entrada de datos es `abcDeFGHi j#`, la salida será 4 (se han introducido cuatro mayúsculas: D ,F ,G ,H)

Finalidad: Mostrar cómo encapsular tareas dentro de funciones y cómo se realiza la llamada entre ellas. Dificultad Baja.

19. Recupere la solución del ejercicio 15 de la Relación de Problemas II (pasar de mayúscula a minúscula y viceversa usando un enumerado) Para que el tipo de dato enumerado sea accesible desde dentro de las funciones, debemos ponerlo antes de definir éstas, es decir, en un ámbito global a todo el fichero. Se pide definir las siguientes funciones y cread un programa principal de ejemplo que las llame:

- a) `Capitalizacion` nos dice si un carácter pasado como parámetro es una minúscula, mayúscula u otro carácter. A dicho parámetro, llamadlo `una_letra`. La función devuelve un dato de tipo enumerado.
- b) `Convierte_a_Mayuscula` comprueba si un carácter pasado como parámetro es minúscula (para ello, debe llamar a la función `Capitalizacion`), 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 `Capitalizacion` podría llamarse igual que el parámetro `caracter` de la función `Convierte_a_Mayuscula`. Esto es porque están en ámbitos distintos y para el compilador son dos variables distintas. Haced el cambio y comprobarlo.

- c) `Convierte_a_Minuscula` análoga a la anterior pero convirtiendo a minúscula. Observad que la constante de amplitud

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

RELACIÓN DE PROBLEMAS III. Funciones y Clases

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

- d) CambiaMayusculaMinuscula, 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 minúscula, devuelve el carácter pasado como argumento.

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

20. *Examen Septiembre 2014.* Dos números amigos son dos números naturales a y b , tales que la suma de los divisores propios de a más uno es igual a b , y viceversa. Un ejemplo de números amigos es el par de naturales (220; 284), ya que:

- Los divisores propios de 220 son 2, 4, 5, 10, 11, 20, 22, 44, 55 y 110, que suman 283, y $283 + 1 = 284$.
- Los divisores propios de 284 son 2, 4, 71 y 142, que suman 219, y $219 + 1 = 220$.

Realice un programa que implemente estas dos tareas:

- a) En primer lugar debe leer dos números naturales e indicar si son o no amigos.
- b) A continuación leerá otro número natural, n , e informará si existe algún número amigo de n en el intervalo centrado en n y de radio 3.

Utilice las funciones que estime oportuno.

Finalidad: Descomponer la solución de un problema en varias funciones. Dificultad Media.

21. Defina una función para implementar la solución del ejercicio 53 de la relación de problemas II (Serie)

Dificultad Media.

22. Defina una función para implementar la solución del ejercicio 57 de la relación de problemas II (número feliz)

Dificultad Media.

Problemas sobre clases

23. En este ejercicio se plantean varias modificaciones. Debe entregar un fichero `cpp` por cada uno de los apartados.

Se desea implementar una clase `Recta` para representar una recta en el plano. Una recta viene determinada por tres coeficientes `A`, `B`, `C`, de forma que todos los puntos (x, y) que pertenecen a la recta verifican lo siguiente (*ecuación general de la recta*):

$$Ax + By + C = 0$$

a) *Definición de la clase y creación de objetos*

Defina la clase `Recta`. En este apartado utilice únicamente datos miembro públicos. Cree un programa principal que haga lo siguiente:

- Defina dos objetos de la clase `Recta`.
- Lea seis reales desde teclado.
- Le asigne los tres primeros a los coeficientes de una recta y los otros tres a la segunda recta.
- Calcule e imprima la pendiente de cada recta aplicando la fórmula:
$$\text{pendiente} = - A / B$$

b) *Métodos públicos*

En vez de calcular la pendiente en el programa principal, vamos a ponerlo como un método de la clase y así lo reutilizaremos todas las veces que necesitemos. Añada un método para el cálculo de la pendiente y modifique el `main` para tener en cuenta este cambio.

¿Añadimos `pendiente` como dato miembro de la recta? La respuesta es que no ¿Por qué?

Añada también los siguientes métodos:

- Obtener la ordenada (`y`) dado un valor de abscisa `x`, aplicando la fórmula:
$$(-C -xA) / B$$
- Obtener la abscisa (`x`) dado un valor de ordenada `y`, aplicando la fórmula:
$$(-C -yB) / A$$

En la función `main` lea un valor de abscisa e imprima la ordenada según la recta. A continuación lea un valor de ordenada e imprima la abscisa que le corresponde. Hágalo sólo con la primera recta.

c) *Datos miembro privados*

Cambie ahora los datos miembro públicos y póngalos privados. Tendrá que añadir métodos para asignar y ver los valores de los datos miembro. Añada métodos

para asignar un valor a cada uno de los tres datos miembro. Modifique el `main` para tener en cuenta estos cambios.

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

IMPORTANT

d) *Política de acceso a los datos miembros*

En vez de usar un método para asignar un valor a cada dato miembro, defina un único método `SetCoeficientes` para asignar los tres a la misma vez.

Observe que los métodos permiten definir la política de acceso a los datos miembro. Si tengo previsto cambiar por separado los coeficientes de la recta, usaré métodos de asignación individuales. En caso contrario, usaré un único método que modifique a la misma vez todos los datos miembro. Incluso pueden dejarse en la clase ambos tipos de métodos para que así el cliente de la clase pueda usar los que estime oportunos en cada momento. Por ahora, mantenga únicamente el método de asignación *en bloque* `SetCoeficientes`.

e) *Constructor*

Modifique el programa principal del último apartado e imprima los valores de los datos miembros de una recta, **antes** de asignarles los coeficientes. Mostrará, obviamente, un valor indeterminado. Para evitar este problema, añada un constructor a la recta para que el objeto esté en un estado válido en el mismo momento de su definición. El constructor deberá tener como parámetros, obligatoriamente, los tres coeficientes de la recta. Tendrá que modificar convenientemente el `main` para tener en cuenta este cambio.

f) *Política de acceso a los datos miembro*

Suprima ahora el método `SetCoeficientes`. De esta forma, una vez creado el objeto (pasándole los datos apropiados en el constructor) ya no podremos modificar los datos miembro. Esto es útil en aquellas situaciones en las que no queremos permitir que el estado del objeto cambie, una vez se ha creado.

g) *Métodos privados*

Vuelva a recuperar el método `SetCoeficientes`. Añada un método privado que nos indique si los coeficientes son correctos, es decir, A y B no pueden ser simultáneamente nulos. Llame a este método donde sea necesario.

Finalidad: Familiarizarnos con la definición de clases. Dificultad Baja.

24. Considere la siguiente definición de la clase `Cronometro`:

```
#include <chrono>
```

```
class Cronometro{  
private:
```

```
typedef std::chrono::time_point<std::chrono::steady_clock>
    Punto_en_el_Tiempo;
typedef chrono::duration <double, nano> IntervaloTiempo;

Punto_en_el_Tiempo inicio;
Punto_en_el_Tiempo final;
public:
    void Reset(){
        inicio = chrono::steady_clock::now();
    }
    double NanoSegundosTranscurridos(){
        final = chrono::steady_clock::now();
        IntervaloTiempo diferencia = final - inicio;

        return diferencia.count();
    }
};
```

No hace falta que entienda el código de la clase sino únicamente cómo utilizar sus métodos públicos. Sirve para medir el tiempo de ejecución de un conjunto de instrucciones. Para ello, basta crear un objeto de esta clase y justo antes del conjunto de instrucciones que queramos cronometrar, debemos ejecutar el método `Reset`. Justo después de las instrucciones, llamaremos al método `NanoSegundosTranscurridos` para saber el número de nanosegundos transcurridos. El cronómetro seguirá en marcha (por lo que podremos llamar al método `NanoSegundosTranscurridos` tantas veces como queramos) hasta que se resetee de nuevo con el método `Reset`.

Defina el método `MiliSegundosTranscurridos` para saber cuántos milisegundos han transcurrido. Este método debe llamar al anterior.

Para realizar una prueba de esta clase, recupere la solución al ejercicio 6 (sumatoria de factoriales). Modifique el programa para que realice la siguiente sumatoria:

```
suma_factoriales = 0;

for (int i=0; i < 2e+7; i++)
    suma_factoriales = suma_factoriales + SumaFactoriales(tope);
```

Ejemplo de entrada: (tope) 20 -- Salida correcta: 53608145800000000

Utilice ahora la clase anterior para realizar una comparación en el tiempo de ejecución entre las versiones ineficiente y eficiente de la función `SumaFactoriales`.

Finalidad: Enfatizar la importancia de la ocultación de información y de la interfaz pública de una clase. Dificultad Baja.

25. Considere la siguiente definición de la clase GeneradorAleatorioEnteros:

```
#include <random> // -> generación de números pseudoaleatorios
#include <chrono> // -> para la semilla

class GeneradorAleatorioEnteros{
private:
    mt19937 generador_mersenne; // Mersenne twister
    uniform_int_distribution<int> distribucion_uniforme;

    long long Nanosec(){
        return chrono::high_resolution_clock::now().
            time_since_epoch().count();
    }
public:
    GeneradorAleatorioEnteros()
        :GeneradorAleatorioEnteros(0, 1){
    }
    GeneradorAleatorioEnteros(int min, int max){
        const int A_DESCARTAR = 70000;
        // ACM TOMS Volume 32 Issue 1, March 2006

        auto semilla = Nanosec();
        generador_mersenne.seed(semilla);
        generador_mersenne.discard(A_DESCARTAR);
        distribucion_uniforme =
            uniform_int_distribution<int> (min, max);
    }

    int Siguiente(){
        return distribucion_uniforme(generador_mersenne);
    }
};
```

No hace falta que entienda el código de la clase sino únicamente cómo utilizar sus métodos públicos. Sirve para generar números aleatorios enteros en un rango de valores.

Esta clase tiene dos constructores. Uno con dos parámetros, `min` y `max` que delimitan el rango correspondiente. El otro constructor no tiene parámetros y establece que únicamente se van a generar ceros y unos (este segundo constructor llama al anterior). Cada vez que se llame al método `Siguiente`, éste devolverá un valor aleatorio en el rango especificado.

RELACIÓN DE PROBLEMAS III. Funciones y Clases

Utilice dicha clase para crear un conjunto de datos de prueba para el ejercicio 11 de esta relación de problemas (centro de atención telefónica) Genere un total de 1000 casos, con un código de empleado entre 1 y 3, un número de segundos de atención entre 40 y 300, un código 0 o 1 de caso resuelto y un grado de satisfacción del cliente entre 0 y 5.

Nota: Realmente, los números generados son *pseudoaleatorios* (puede consultar Internet para tener una idea de este concepto) En este ejemplo se han generado según una distribución uniforme pero se pueden generar números con las probabilidades dadas por muchas otras distribuciones -disponibles en la biblioteca `random` de C++11-.

Finalidad: Enfatizar la importancia de la ocultación de información y de la interfaz pública de una clase. Dificultad Baja.

26. Defina la clase `CalculadoraNotaFebrero` para calcular la nota en la convocatoria ordinaria de Febrero de un alumno en la asignatura de Fundamentos de Programación. Para ello, debe considerar lo siguiente:

- Cada alumno es calificado con 4 notas (especificadas de 0 a 10): evaluación continua, dos exámenes prácticos y un examen escrito. Por defecto, la ponderación de cada parte en el cómputo de la nota final es 10 %, 10 %, 20 % y 60 % respectivamente.

Estos porcentajes son los mismos para todos los grupos. En cualquier caso, se quiere contemplar la posibilidad de manejar otros distintos.

- El profesor de cada grupo, tiene la posibilidad, si así lo desea, de subir la nota del examen escrito un máximo de 0,5 puntos. Esta subida sólo se aplica a aquellos alumnos para los que, después de aplicarla, obtienen una nota mayor o igual que 5.

- Para poder aprobar la asignatura, es preciso haber sacado al menos un 4 en la nota del examen escrito. Este límite (4), al igual que los porcentajes del primer apartado, es el mismo para todos los grupos y se aplica antes de la subida de nota especificada en el apartado anterior.

Si el alumno no supera la nota mínima de 4 en el examen escrito, la nota final será la nota del examen escrito.

Construya un programa principal que lea los datos en el siguiente orden y calcule la nota final de cada alumno. No hay límite en el número de grupos que se van a introducir.

0.1 0.1 0.2 0.6	-> Las ponderaciones de las 4 partes Común a todos los grupos Siempre se introducirán al menos estos 4 datos
0.5	-> Subida lineal del grupo 1

RELACIÓN DE PROBLEMAS III. Funciones y Clases

```
2.5 3.5 7.5 4.4    -> Notas del alumno 1 del grupo 1
6.4 9.5 8.5 7.2    -> Notas del alumno 2 del grupo 1
.....
-1                  -> Fin de datos del grupo 1
0.3                -> Subida lineal del grupo 2
3.5 6.4 5.5 6.4    -> Notas del alumno 1 del grupo 2
1.4 2.5 3.4 1.3    -> Notas del alumno 2 del grupo 2
.....
-1                  -> Fin de datos del grupo 2
-1                  -> Fin de datos
```

En `decsai` se encuentra un fichero de prueba para este ejercicio, así como un fichero con las notas finales correspondientes.

Ejemplo de entrada:

```
0.1 0.1 0.2 0.6
0.5
7 7 7 3.9
5 5 5 4.4
5 5 5 4.5
-1
0.4
7 7 7 3.9
5 5 5 4.4
5 5 5 4.5
5 5 5 5
-1 -1
```

-- Salida correcta:

```
3.9 4.6 5
3.9 4.6 4.7 5.2
```

Finalidad: Diseño de una clase. Dificultad Media.

27. Recupere la solución del ejercicio 8 de esta relación de problemas sobre la función gaussiana. En vez de trabajar con funciones, plantee la solución con una clase. Debe diseñar la clase teniendo en cuenta que la función matemática gaussiana viene determinada únicamente por el valor de la esperanza y la desviación, es decir, son estos dos parámetros lo que distinguen a una función gaussiana de otra.

Finalidad: Diseño de una clase. Dificultad Baja.

28. Recupere la solución del ejercicio 10 de esta relación de problemas (población con funciones). Re-escríbalo para que los cómputos relacionados con la población estén encapsulados en una clase. La lectura de los valores en los rangos adecuados se hará

RELACIÓN DE PROBLEMAS III. Funciones y Clases

con las mismas funciones que ya se definieron en ese ejercicio. Modifique apropiadamente el programa principal.

Finalidad: Diseño de una clase. Dificultad Baja.

29. Recupere la solución del ejercicio 9 de esta relación de problemas (parking con funciones). Re-escríbalo para que los cómputos relacionados con el cálculo de la tarifa, estén encapsulados en una clase. Mantenga la definición de la función `MinutosEntreInstantes` tal y como está. Modifique apropiadamente el programa principal.

Finalidad: Diseño de una clase. Dificultad Media.

30. En el ejercicio 16 de esta relación de problemas se definieron varias funciones para operar sobre una progresión geométrica. Definid ahora una clase para representar una progresión geométrica.

- a) Diseñad la clase pensando cuáles serían los datos miembro *esenciales* que definen una progresión geométrica, así como el constructor de la clase.
- b) Definir un método `Termino` que devuelva el término k -ésimo.
- c) Definid los métodos `SumaHastaInfinito`, `SumaHasta`, `MultiplicaHasta`.
- d) Cread un programa principal que lea los datos miembro de una progresión, cree el objeto correspondiente y a continuación lea un entero `tope` e imprima los `tope` primeros términos de la progresión, así como la suma hasta `tope` de dichos términos.

Finalidad: Comparar la ventaja de un diseño con clases a uno con funciones. Dificultad Baja.

31. Se quiere construir una clase `DepositoSimulacion` para simular préstamos, ofreciendo la funcionalidad descrita en los ejercicios 20 (reinvierte capital e interés un número de años) y 21 (reinvierte capital e interés hasta obtener el doble de la cantidad inicial) de la relación de problemas II (página RP-II.8). Por tanto, la clase debe proporcionar, para un capital y unos intereses dados, métodos para:

- a) Calcular el capital que se obtendrá al cabo de un número de años,
- b) 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:

- ¿Cuáles son sus datos miembro? Parece claro que el capital y el interés sí lo serán ya que cualquier operación que se nos ocurra hacer con un objeto de la clase `DepositoSimulacion` involucra a ambas cantidades. ¿Pero y el número de años?

- ¿Qué constructor definimos?
- ¿Queremos modificar el capital y el interés una vez creado el objeto?
- ¿Queremos poder modificarlos de forma independiente?
- ¿Hay alguna restricción a la hora de asignar un valor al capital e interés?
- ¿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.

32. Recupere la solución del ejercicio 15 de esta relación de problemas (cómputo del salario en función de las horas trabajadas) Defina una clase Nomina para gestionar el cómputo del salario final. Suponga que el porcentaje de incremento en la cuantía de las horas extras (50 %) y el número de horas que no se tarifan como extra (40) son valores que podrían cambiar, aunque no de forma continua. El número de horas trabajadas y la cuantía a la que se paga cada hora extraordinaria, sí son cantidades que varían de un trabajador a otro.

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

33. Recuperad la solución del ejercicio 17 (actualización de la retención fiscal) de la relación de problemas II. En este problema se leían caracteres de teclado ('s'/'n') para saber si una persona era autónomo, pensionista, etc.

```
cout << "\n¿La persona es un trabajador autónomo? (s/n) " ;  
do{  
    cin >> opcion;  
    opcion = toupper(opcion);  
}while (opcion != 'S' && opcion != 'N');
```

Este código era casi idéntico para la lectura del resto de los datos. Para evitarlo, definid una clase MenúSiNO que encapsule esta funcionalidad y cambiar el programa principal para que use esta clase.

34. Recuperad la solución del ejercicio 23 (recta) de esta relación de problemas. Se pide crear un programa principal que haga lo siguiente:

- Se presentará al usuario un menú principal para salir del programa o para introducir los valores de los coeficientes A, B, C de la recta.
- Una vez introducidos los coeficientes se presentará al usuario un segundo menú, para que elija alguna de las siguientes opciones:
 - Mostrar el valor de la pendiente de la recta.

- Mostrar la ordenada dada una abscisa (el programa tendrá que pedir la abscisa)
- Mostrar la abscisa dada una ordenada (el programa tendrá que pedir la ordenada)
- Volver al menú principal.

Para resolver este problema, debe crear dos clases `MenuPrincipal` y `MenuOperaciones`.

Finalidad: Trabajar con varias clases en un programa. Dificultad Media.

35. Se quiere construir una clase `Nomina` para realizar la funcionalidad descrita en el ejercicio 18 de la relación de problemas I sobre la nómina del fabricante y diseñador (página RP-I.7). Cread los siguientes programas (entregad un fichero por cada uno de los apartados):

- a) Suponed que sólo gestionamos la nómina de una empresa en la que hay un fabricante y tres diseñadores. Los salarios brutos se obtienen al repartir los ingresos de la empresa, de forma que el diseñador cobra el doble de cada fabricante.

El programa leerá el valor de los ingresos totales y calculará los salarios brutos de los fabricantes y diseñador, llamando a los métodos oportunos de la clase `Nomina`.

- b) Supongamos que se aplica una retención fiscal y que ésta es la misma para los fabricantes y el diseñador. En el constructor se establecerá el porcentaje de retención fiscal (de tipo `double`) y posteriormente no se permitirá que cambie, de forma que todas las operaciones que se hagan serán siempre usando la misma retención fiscal. Los salarios netos se obtienen al aplicar la retención fiscal a los salarios brutos (después de repartir los ingresos totales de la empresa):

```
salario_neto = salario_bruto -  
              salario_bruto * retencion_fiscal / 100.0
```

El programa leerá el valor de los ingresos totales y la retención fiscal a aplicar y calculará los salarios brutos y netos de los fabricantes y diseñador, llamando a los métodos oportunos de la clase `Nomina`.

- c) Supongamos que gestionamos las nóminas de varias sucursales de una empresa. Queremos crear objetos de la clase `Nomina` que se adapten a las características de cada sucursal:

- En cada sucursal hay un único diseñador pero el número de fabricantes es distinto en cada sucursal. Por tanto, el número de fabricantes habrá que especificarlo en el constructor y posteriormente no podrá cambiarse.
- La forma de repartir el dinero es la siguiente: el diseñador se lleva una parte del total y el resto se reparte a partes iguales entre los fabricantes. En los apartados anteriores, por ejemplo, la parte que se llevaba el diseñador era

2/5 y el resto (3/5) se repartía entre los tres fabricantes. La parte que el diseñador se lleva puede ser distinta entre las distintas sucursales (2/5, 1/6, etc), pero no cambia nunca dentro de una misma sucursal. Por tanto, el porcentaje de ganancia (2/5, 1/6, etc) habrá que especificarlo en el constructor y posteriormente no podrá cambiarse.

- Las retenciones fiscales de los fabricantes y diseñador son distintas. Además, se prevé que éstas puedan ir cambiando durante la ejecución del programa. Por lo tanto, no se incluirán como parámetros en el constructor.

El programa leerá los siguientes datos desde un fichero externo:

- El número de sucursales.
- Los siguientes valores por cada una de las sucursales:
 - Ingresos totales a repartir
 - Número de fabricantes
 - Parte que se lleva el diseñador
 - Retención fiscal del diseñador
 - Retención fiscal de los fabricantes

Por ejemplo, el siguiente fichero indica que hay dos sucursales. La primera tiene unos ingresos de 300 euros, 3 fabricantes, el diseñador se lleva 1/6, la retención del diseñador es del 20 % y la de cada fabricante un 18 %. Los datos para la segunda son 400 euros, 5 fabricantes, 1/4, 22 % y 19 %.

2
300 3 6 20 18
400 5 4 22 19

El programa tendrá que imprimir los salarios brutos y netos del diseñador y de los fabricantes por cada una de las sucursales, llamando a los métodos oportunos de la clase *Nomina*.

Finalidad: Diseño de una clase y trabajar con datos miembro constantes. Dificultad Media.

36. Se quiere construir una clase para representar la tracción de una bicicleta, es decir, el conjunto de estrella (engranaje delantero), cadena y piñón (engranaje trasero). Supondremos que la estrella tiene tres posiciones (numeradas de 1 a 3, siendo 1 la estrella más pequeña) y el piñón siete (numeradas de 1 a 7, siendo 1 el piñón más grande). La posición inicial de marcha es estrella = 1 y piñón = 1.

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 saltos de uno o de dos. Si ha llegado al límite superior (inferior) y se llama al método para subir (bajar) la estrella, la posición de ésta no variará. Lo mismo se aplica al piñón.

Cread un programa principal que lea desde un fichero externo los movimientos realizados e imprima la situación final de la estrella y piñón. Los datos se leerán en el

siguiente formato: tipo de plato (piñón o estrella) seguido del tipo de movimiento. Para codificar esta información se usarán las siguientes letras: E indica una estrella, P un piñón, S para subir una posición, B para bajar una posición, T para subir dos posiciones y C para bajar dos posiciones. T y C sólo se aplicarán sobre los piñones.

E S P S P S P S P C E S E B #

En este ejemplo los movimientos serían: la estrella sube, el piñón sube en tres ocasiones sucesivas, el piñón baja dos posiciones de golpe, la estrella sube y vuelve a bajar. Supondremos siempre que la posición inicial de la estrella es 1 y la del piñón 1. Así pues, la posición final será Estrella=1 y Piñón=2.

Mejorad la clase 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:

- Estrella igual a 1 y piñón mayor o igual que 5
- Estrella igual a 2 y piñón o bien igual a 1 o bien igual a 7
- Estrella igual a 3 y piñón menor o igual que 3

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

37. Recuperad la solución del ejercicio 36 de la Relación de Problemas II (Empresa). Reescribid el programa principal usando una clase Ventas para gestionar los cómputos de las ventas realizadas. Únicamente se pide que se indiquen las cabeceras de los métodos públicos de la clase y las llamadas a éstos en el programa principal. No hay que implementar ninguno de los métodos.

Debe suponer que la clase gestionará las ventas de exactamente tres sucursales. Los códigos de dichas sucursales son enteros cualesquiera (no necesariamente 1, 2, 3, como ocurría en el ejercicio 36 de la Relación de Problemas II)

El programa principal sería de la siguiente forma:

```
Ventas ventas_empresa;  
.....  
while (identif_sucursal != TERMINADOR){  
    cin >> cod_producto;  
    cin >> unidades_vendidas;  
  
    --> Actualiza el número de unidades  
        vendidas de la sucursal leída  
        llamando a un método de ventas_empresa  
  
    cin >> identif_sucursal;  
}
```

--> Obtener el identificador y el número de ventas de la sucursal ganadora llamando a un método de ventas_empresa

Finalidad: Diseño de una clase. Dificultad Media.

38. Implementar los métodos de la clase Ventas del ejercicio anterior.

Finalidad: Diseño de una clase. Dificultad Media.

39. Implemente una clase para representar un número complejo. Un complejo se define como un par ordenado de números reales (a, b) , donde a representa la parte real y b la parte imaginaria. Construya un programa principal que lea la parte real e imaginaria, cree el objeto e imprima el complejo en la forma $a + bi$.

Por ahora no podemos implementar métodos para sumar, por ejemplo, dos complejos. Lo veremos en el último tema.

40. 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.

41. Implementad la clase del ejercicio 40 de esta relación de problemas. *Dificultad Media.*
42. Definid una clase Dinero para poder trabajar de forma precisa con datos monetarios. La clase tendrá dos datos miembro, euros y centimos y cuando se modifiquen éstos, la clase debe permitir que se introduzca un número de céntimos mayor de 100. Por ejemplo, si asignamos 20 euros y 115 céntimos, el objeto debe almacenar 21

en euros y 15 en centimos. En el último tema veremos cómo sumar o restar dos objetos de la clase Dinero. Incluid un sencillo programa principal que llame a los métodos.

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

43. Recuperad la solución del ejercicio 38 (Empresa) y modifcadlo convenientemente para que los datos miembros que referencia los identificadores de las sucursales sean constantes.

Finalidad: Trabajar con datos miembros constantes. Dificultad Baja.

44. 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**.

Cread un programa principal que pida algunos datos y muestre los valores convertidos.

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.

RELACIÓN DE PROBLEMAS III. Funciones y Clases

45. Construid una clase llamada **MedidaAngulo** que represente una medida de un ángulo. Al igual que se hizo en el ejercicio 44, 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.

46. Recupere la solución del ejercicio 29 (Parking con una clase) Defina un **struct** llamado **InstanteTiempo** para almacenar la hora, minutos y segundos que constituyen un instante de tiempo. Cambie la definición de la función **MinutosEntreInstantes** y el programa principal para que trabaje con este tipo **struct**.

Finalidad: Trabajar con funciones y el tipo struct. Dificultad Baja.

47. Defina un **struct** llamado **CoordenadasPunto2D** para representar un par de valores reales correspondientes a un punto en \mathbb{R}^2 .

Defina una función **DistanciaEuclidea** para que calcule la distancia entre dos puntos cualesquiera. Cree un programa principal que vaya leyendo 4 valores reales desde teclado representando las coordenadas de dos puntos y calcule la distancia euclídea entre ellos. Cada vez que se lean los cuatro valores se le preguntará al usuario si quiere seguir introduciendo datos o no (con las opciones '**s**'/'**n**').

Ejemplo de entrada:

s 3.1 4.2 5.3 6.4 **j** **k** **s** 2.1 4.9 -3.2 0 **s** 1 5 1 5 **n**

-- Salida correcta: 3.11127 7.21803 0

Finalidad: Trabajar con funciones y el tipo struct. Dificultad Baja.

48. Recupere la definición del registro **CoordenadasPunto2D** del ejercicio 47 de esta relación de problemas y la solución al ejercicio 3 (Circunferencia) de la relación de problemas I.

Cree ahora una clase llamada **Circunferencia**. Para establecer el centro, se usará un dato miembro que ha de ser de tipo **CoordenadasPunto2D**.

Añada métodos para obtener la longitud de la circunferencia y el área del círculo interior.

Añada también un método para saber si la circunferencia contiene a un punto cualquiera. 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:

$$(x_0 - x_1)^2 + (y_0 - y_1)^2 \leq r^2$$

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

RELACIÓN DE PROBLEMAS III. Funciones y Clases

Cree un programa principal que lea el centro y el radio de una circunferencia, las coordenadas de un punto y muestre en pantalla la longitud de la circunferencia, el área del círculo y nos diga si el punto está dentro o no de la circunferencia.

Ejemplo de entrada: 2.1 3.2 5.8 2.2 4.6

-- Salida correcta: 36.4425 105.683 El punto está dentro

Ejemplo de entrada: 2.1 3.2 5.8 2.2 10.36

-- Salida correcta: 36.4425 105.683 El punto no está dentro

Finalidad: Trabajar con clases y el tipo struct. Dificultad Baja.

RELACIÓN DE PROBLEMAS IV. Vectores

En los ejercicios que pida trabajar sobre la clase **SecuenciaCaracteres**, use la siguiente definición:

SecuenciaCaracteres	
- const int	TAMANIO
- char	vector_privado[TAMANIO]
- int	total_utilizados
- void	IntercambiaComponentes_en_Posiciones(int pos_izda, int pos_dcha)
+	SecuenciaCaracteres()
+ int	TotalUtilizados()
+ int	Capacidad()
+ void	Aniade(char nuevo)
+ void	Modifica(int pos_a_modificar, char valor_nuevo)
+ char	Elemento(int indice)
+ void	EliminaTodos()
+ int	PrimeraOcurrenciaEntre(int pos_izda, int pos_dcha, char buscado)
+ int	PrimeraOcurrencia(char buscado)
+ int	BusquedaBinaria(char buscado)
+ int	PosicionMinimoEntre(int izda, int dcha)
+ int	PosicionMinimo()
+ void	Inserta(int pos_insercion, char valor_nuevo)
+ void	Elimina(int pos_a_eliminar)
+ void	Ordena_por_Seleccion()
+ void	Ordena_por_Insercion()
+ void	Ordena_por_Burbuja()
+ void	Ordena_por_BurbujaMejorado()

Puede encontrar el código en la dirección siguiente:

http://decsai.ugr.es/~carlos/FP/IV_clase_secuencia_caracteres.cpp

En todos los ejercicios, tenga en cuenta lo siguiente:

- Siempre ha de diseñar una batería de pruebas, que intente garantizar que la solución propuesta funcione correctamente en cualquier situación extrema.
- Recuerde lo visto en las transparencias del primer tema. Si queremos leer datos de tipo `char`, para poder leer un espacio en blanco **no** puede emplear `cin >> caracter`, sino `caracter = cin.get()`. Cada vez que se ejecute `cin.get()` el compilador lee un carácter (incluido el espacio en blanco, el tabulador y el retorno de carro '`\n`') desde la entrada de datos por defecto. En definitiva, el bucle de lectura de datos será del tipo:

```
caracter = cin.get();  
  
while (caracter != TERMINADOR){  
    .....  
    caracter = cin.get();  
}
```

Cuando la entrada de datos es desde un fichero de texto, cada ejecución de `cin.get()` lee directamente un carácter (incluidos los espacios en blanco, tabuladores y retornos de carro) y el programa pasa a la siguiente sentencia.

Si fuese desde el teclado, habría que esperar a que el usuario introdujese el retorno de carro para que los datos pasasen al buffer y una vez ahí, se ejecutarían automáticamente todos los `cin.get()` (recordad lo visto al final del primer tema). En el caso de que, por ejemplo, quisiéramos parar la lectura cuando se hubiesen introducido más de un número tope de caracteres, no podríamos hacerlo con la lectura desde teclado ya que los datos no pasan al buffer hasta que no se pulse el retorno de carro.

- Algunos ejercicios usan un vector de caracteres muy grande, como para albergar, por ejemplo, el texto del Quijote disponible en

<http://decsai.ugr.es/~carlos/FP/Quijote.txt>

Para que el vector con dicho texto quepa en la pila, debe hacer los siguientes cambios:

- Declare una constante de tamaño del vector que permita almacenar todos los caracteres.

```
const int TAMANIO = 2500000;  
char texto[TAMANIO];
```

Si se prefiere, para que quede más claro el número de caracteres reservados en memoria, puede usar una constante `double` en formato científico y dejar que C++ haga el casting a `int` automáticamente:

```
const int TAMANIO = 25e+5; // int = double  
char texto[TAMANIO];
```

En el caso de que trabaje dentro de una clase (por ejemplo `SecuenciaCaracteres`), debe usar una constante estática:

```
class SecuenciaCaracteres{
private:
    static const int TAMANIO = 25e+5;
    char vector_privado[TAMANIO];
```

- Aumente el tamaño de la pila asignada por el sistema operativo al programa generado por el compilador. Para ello, debe seleccionar desde DevC++:

Herramientas -> Opciones del Compilador ->

Señale la opción

Añadir los siguientes comandos al llamar al compilador
y en la caja de texto introduzca lo siguiente (todo seguido sin espacios en blanco):

`-Wl,--stack,2600000`

El último número es el número de bytes que va a permitir almacenar en la pila. En nuestro ejemplo, teníamos 2.5 millones de caracteres, por lo que necesitaremos 2.5 MB. Reservamos algo más para otras variables del programa: en total 2.6 millones de bytes (2.6 MB)

Tenga en cuenta que aquí no puede usar la notación científica.

Problemas con vectores declarados en el main

En este conjunto de ejercicios, se trabajará con un vector declarado directamente en el `main`. Será un vector sin huecos y delimitaremos el final con una variable entera `total_utilizados` que indique el número de componentes utilizadas.

1. [Palíndromo e invierte]. Tenga en cuenta la observación al inicio de esta relación de problemas sobre la lectura de los caracteres. Para poder leer caracteres, incluyendo los espacios en blanco, hay que usar `caracter = cin.get()`, en vez de `cin >> caracter`.

Implemente algoritmos para realizar las siguientes tareas:

- a) Comprobar si el vector es un palíndromo, es decir, que se lee igual de izquierda a derecha que de derecha a izquierda. Por ejemplo, `{'a', 'b', 'b', 'a'}` sería un palíndromo, pero `{'a', 'c', 'b', 'a'}` no lo sería. Si la secuencia tiene un número impar de componentes, la que ocupa la posición central es descartada, por lo que `{'a', 'b', 'j', 'b', 'a'}` sería un palíndromo.
- b) Invertir el vector. Si éste contenía, por ejemplo, los caracteres `{'m', 'u', 'n', 'd', 'o'}`, después de llamar al método se quedará con `{'o', 'd', 'n', 'u', 'm'}`.

Construya un programa principal y declare un vector de caracteres de tamaño 100. Lea las componentes considerando como terminador el carácter # (éste no forma parte de la secuencia) y que no se introduzcan más de 100 caracteres. A continuación, el programa debe determinar si la secuencia es un palíndromo. En caso negativo, debe invertirla y mostrar el resultado en pantalla.

Ejemplo de entrada: a# — Salida correcta: Es un palíndromo

Ejemplo de entrada: abcba# — Salida correcta: Es un palíndromo

Ejemplo de entrada: abccba# — Salida correcta: Es un palíndromo

Ejemplo de entrada: abcdab#

— Salida correcta: No es un palíndromo. Secuencia invertida: abdcba

Finalidad: Recorrer las componentes de un vector. Dificultad Baja.

2. [k mayores que otro valor]. ([Examen Septiembre 2016](#)) Se dispone de un conjunto de reales positivos y se quiere construir otro vector `mayores_que` que contenga los k primeros valores que son mayores o iguales que otro valor de referencia. Los valores que se obtengan como resultado deben estar ordenados de menor a mayor.

Por ejemplo, si el vector original es

`{5.2 2.5 7.3 4.2 3.1 4.9}`

y el valor de referencia es 4.3, el vector `mayores_que` debe quedar así:

{4.9 5.2 7.3}

Construya un programa que vaya leyendo datos desde teclado e introduzcalos en un vector. La entrada de datos termina con el -1. A continuación lea el real de referencia y el entero k . Calcule el vector `mayores_que` e imprímalo en pantalla.

Aplique el siguiente algoritmo:

Copiar el vector original en "mayores_que"

Ordenar el vector "mayores_que" de menor a mayor
(utilizar el algoritmo de ordenación por inserción)

Seleccionar los k primeros que sean mayores que la referencia

Utilice también el fichero de datos

http://decsai.ugr.es/~carlos/FP/mayores_que.txt

La salida correcta se encuentra en el fichero

http://decsai.ugr.es/~carlos/FP/mayores_que_solucion.txt

Finalidad: Ordenación de un vector. Dificultad Baja.

3. [k mayores que otro valor, eficiente]. Modifique la solución del ejercicio 2 usando un algoritmo más eficiente. Observe que no hace falta ordenar todo el vector, sino únicamente considerar los datos que son mayores que la referencia.

Aplique el siguiente algoritmo:

Recorrer las componentes del vector original

Si es mayor que la referencia, insertar dicho valor de forma ordenada en el vector "mayores_que"

El vector "mayores_que" siempre tendrá,
como mucho, k componentes

Mientras que la versión vista en el ejercicio 2 tardaba varios segundos, esta nueva versión tarda menos de un segundo.

Finalidad: Recorrido sobre un vector, insertando componentes. Dificultad Media.

4. [Moda]. Se quiere calcular la moda de un vector de caracteres, es decir, el carácter que más veces se repite. Por ejemplo, si el vector fuese

{'l', 'o', 's', ' ', 'd', 'o', 's', ' ', 'c', 'o', 'f', 'r', 'e', 's'}

los caracteres que más se repiten son 'o' y 's' con un total de 3 apariciones. La moda sería cualquiera de ellos, por ejemplo, el primero encontrado 'o'.

Para almacenar de forma conjunta el carácter y la frecuencia usaremos el siguiente struct:

```
struct FrecuenciaCaracter{  
    char caracter;  
    int frecuencia;  
}
```

El campo **caracter** contendrá el carácter en cuestión ('o') y en el campo **frecuencia** el conteo de la moda (3).

Construya un programa que lea caracteres con `cin.get()` (repase lo visto al inicio de esta relación de problemas) hasta que el usuario introduzca el carácter #. Almacene todos los valores en un vector de caracteres. A continuación, calcule la moda y muéstrela en pantalla junto con su frecuencia.

Para calcular la moda, se recomienda que use un vector auxiliar en el que almacene los caracteres que ya se han procesado en algún momento anterior.

Utilice el texto del Quijote sin espacios en blanco disponible en:

http://decsai.ugr.es/~carlos/FP/Quijote_sin_espacios.txt

La moda es la letra 'e' con un total de 217900 apariciones.

Finalidad: Recorridos sobre un vector. Dificultad Media.

5. [Descodifica]. (*Examen Febrero 2016*) Dado un vector de caracteres que contiene un mensaje cifrado, se pide construir otro vector nuevo con el mensaje descifrado. La forma de descifrado consiste en coger la primera y última letra de cada palabra. Las palabras están separadas por uno o más espacios en blanco o el final del vector. Si la última palabra no tiene espacios en blanco a su derecha, se coge sólo el primer carácter.

Por ejemplo, si denotamos el inicio y final de la secuencia con un corchete, entonces:

[Hidrógeno limpia] se descodificaría como [Hola]

Reserve memoria para trabajar con un máximo de 1000 caracteres.

Para leer el mensaje cifrado debe leer caracteres con `cin.get()` (repase lo visto al inicio de esta relación de problemas) hasta que el usuario introduzca el carácter #. A continuación, el programa mostrará la cadena descodificada.

Ejemplo de entrada: [Hidrógeno limpia] -- Salida correcta: [Hola]

Ejemplo de entrada: [Hidrógeno limpia] -- Salida correcta: [Hol]

Ejemplo de entrada: [Hidrógeno] -- Salida correcta: [H]

Ejemplo de entrada: [Hidrógeno] -- Salida correcta: [Ho]

Ejemplo de entrada: [H] -- Salida correcta: [H]

Ejemplo de entrada: [H] -- Salida correcta: [H]

Finalidad: Recorrido de las componentes de un vector, controlando qué ha ocurrido anteriormente. Dificultad Media.

6. [Pinta dígitos]. Construya la función con cabecera:

```
string Digitos (int n)
```

para que extraiga en un **string** los dígitos del número **n** tal y como se indica en el ejercicio **30** de la relación de problemas II.

Finalidad: Trabajar con la clase string. Dificultad Baja.

7. [top k]. (*Examen Febrero 2015*) Se dispone de una serie de datos pluviométricos de varios años, en concreto del número de litros que han caído en un día en cada una de las ciudades del mundo. Se quiere calcular los **k** mayores índices pluviométricos, ordenados de mayor a menor. Se pide implementar el siguiente algoritmo:

Construya un programa que vaya leyendo datos desde teclado hasta que se introduzca -1. A continuación lea el número **k** y aplique el siguiente algoritmo:

Ordenar el vector de mayor a menor

(se recomienda usar el algoritmo de ordenación por inserción)

Seleccionar los **k** primeros e introducirlos
en **topk**

Finalmente, imprima los valores del vector **topk** en pantalla.

Ejemplo de entrada: 2.1 0.7 12.4 2.6 3.2 -1 2 -- Salida correcta:
12.4 3.2

Posteriormente se verá una versión mejorada de este ejercicio. *Finalidad: Ordenación de un vector. Dificultad Baja.*

Problemas con vectores y clases

8. [Palíndromo e invierte dentro de la clase SecuenciaCaracteres]. Añada los métodos **EsPalindromo**, **Invierte** a la clase **SecuenciaCaracteres** que implementen las tareas descritas en el ejercicio **1** de esta relación de problemas.

Incluya un programa principal de prueba similar al del ejercicio **1**.

Finalidad: Diseñar las cabeceras de los métodos que acceden a las componentes del vector. Dificultad Baja.

9. [Intercambia componentes]. Sobre la clase **SecuenciaCaracteres**, añada el método **IntercambiaComponentes** para intercambiar dos componentes de la secuencia. Por ejemplo, si la secuencia contiene { 'h', 'o', 'l', 'a' }, después de intercambiar las componentes 1 y 3, se quedaría con { 'h', 'a', 'l', 'o' }.

¿Qué debería hacer este método si los índices no son correctos?

Modifique la implementación del método **Invierte** del ejercicio 8, para que lo haga llamando a **IntercambiaComponentes**.

Imprima las componentes de la secuencia desde el **main**, antes y después de llamar a dicho método. Para ello, use el método **ToString()** de la clase **SecuenciaCaracteres**.

Finalidad: Ejemplo de método que modifica el estado del objeto. Gestión de precondiciones. Dificultad Baja.

10. [Moda]. Recupere la solución del ejercicio 4. Defina el método **Moda** dentro de la clase **SecuenciaCaracteres**. Este método debe devolver un dato de tipo **FrecuenciaCaracter** (struct) Aplíquelo sobre el texto del Quijote sin espacios en blanco.

Finalidad: Diseñar las cabeceras de los métodos que acceden a las componentes del vector. Dificultad Baja.

11. [Elimina ocurrencias]. Sobre la clase **SecuenciaCaracteres**, añada el método **EliminaOcurrencias** para eliminar todas las apariciones de un determinado carácter **a_borrar**. Por ejemplo, después de eliminar el carácter 'o' de la secuencia { 'S', 'o', 'Y', ' ', 'y', 'o' }, ésta debe quedarse con { 'S', 'Y', ' ', 'y' }. Un primer algoritmo para resolver este problema sería el siguiente (en ejercicios posteriores se verán métodos más eficientes):

Recorrer todas las componentes de la secuencia

Si la componente es igual al carácter **a_borrar**, eliminarla
(desplazando hacia la izda las componentes que hay a su dcha)

Queremos implementarlo llamando al método **Elimina** (que borra un único carácter). La implementación de este método se ha visto en clase de teoría.

```
class SecuenciaCaracteres{  
    .....  
    void EliminaOcurrenciasERROR(){  
        for (int i = 0; i < total_utilizados; i++)  
            if (vector_privado[i] == a_borrar)  
                Elimina(i);  
    }  
};
```

El anterior código tiene un fallo. ¿Cuál? Pruébelo con cualquier secuencia que tenga el carácter a borrar dos veces consecutivas. Proponga una solución e impleméntela.

Finalidad: Recorrido sencillo de un vector con dos bucles anidados. Dificultad Baja.

12. [Elimina bloque, versión ineficiente]. Sobre la clase SecuenciaCaracteres, añada el método EliminaBloque para que borre todos los caracteres que haya entre dos posiciones. Por ejemplo, después de eliminar el bloque que hay entre las posiciones 1 y 3 de la secuencia { 'S', 'o', 'Y', ' ', 'y', 'o' }, ésta debe quedarse con { 'S', 'y', 'o' }.

Un primer algoritmo para resolver este problema sería el siguiente (en ejercicios posteriores se verán métodos más eficientes):

Para borrar el bloque entre izda y dcha:

 Recorrer cada componente -i- de la secuencia
 entre las posiciones izda y dcha
 Eliminar dicha componente -i-

Para eliminar una posición llame al método Elimina (que borra un único carácter) La implementación de este método se ha visto en clase de teoría.

Construya un programa que lea caracteres hasta el terminador #. A continuación lea dos enteros que representen las posiciones izquierda (≥ 0) y derecha (entre izquierda y la última componente de la secuencia) (use las funciones LeeIntMayorIgualQue y LeeIntRango de los ejercicios 4 y 5 de esta relación de problemas) e imprima la secuencia resultante de quitar el bloque de caracteres.

Ejemplo de entrada: abcdefg# 1 3 -- Salida correcta: aefg

Ejemplo de entrada: abcdefg# 1 0 -1 15 3 -- Salida correcta: aefg

Ejemplo de entrada: abcdefg# 0 5 -- Salida correcta: g

Finalidad: Recorrido sencillo de un vector con dos bucles anidados. Dificultad Baja.

13. [Elimina bloque, versión eficiente]. Resuelva el ejercicio 12 pero de una forma eficiente. Para ello, implemente el siguiente algoritmo:

Para borrar el bloque entre izda y dcha:

 Calcular num_a_borrar como dcha - izda + 1

 Recorrer las componentes -i- de la secuencia
 entre las posiciones dcha+1 hasta el final
 Colocar la componente -i- en la posición
 i - num_a_borrar

 Actualizar total_usados

Este algoritmo resuelve el problema con un único bucle mientras que la versión ineficiente recurría a dos bucles anidados.

Finalidad: Recorrido sencillo de un vector con dos bucles anidados. Dificultad Baja.

14. [Elimina ocurrencias, versión eficiente]. El algoritmo del ejercicio 11 es muy ineficiente ya que requiere trasladar muchas veces muchas posiciones (usa dos bucles anidados). Para comprobarlo, recupere el texto

http://decsai.ugr.es/~carlos/FP/Quijote_con_ruido.txt

En él aparecen numerosas ocurrencias del carácter '˜' (código ASCII 126) como ruido en el texto y se quieren eliminar. Si aplica el algoritmo ineficiente, la ejecución puede tardar más de *12 minutos*.

Para resolver eficientemente este problema se propone utilizar dos variables, **posicion_lectura** y **posicion_escritura** que nos vayan indicando, en cada momento, la componente que se está leyendo y el sitio dónde tiene que escribirse. Por ejemplo, supongamos que en un determinado momento la variable **posicion_lectura** vale 6 y **posicion_escritura** 3. Si la componente en la posición 6 es el carácter a borrar, simplemente avanzaremos **posicion_lectura**. En caso contrario, la colocaremos en la posición 3 y avanzaremos una posición ambas variables.

Implemente este algoritmo y observe la diferencia de tiempo al ejecutarlo sobre el Quijote, ya que ahora el tiempo de ejecución es de unos *8 milisegundos*.

Como ampliación: si quiere saber con exactitud el tiempo de ejecución, puede utilizar un objeto de la clase **Cronometro** tal y como se vio en el ejercicio 24 de la relación de problemas III.

Finalidad: Modificar un vector a través de dos apuntadores. Dificultad Media.

15. [Parking]. Recupere la solución del ejercicio 9 de la relación de problemas III (párking). Re-escríbalo definiendo la clase **TarifadorParking** para calcular la tarifa.

La clase debe permitir cualquier número de tramos. Para ello, haga lo siguiente:

- Defina dos vectores como datos miembro de la clase. En uno almacenaremos los límites de los tramos y en el otro la correspondiente tarifa.
- Defina el siguiente método:

```
void AniadeTramo(double limite_superior_tramo,  
                  double tarifa_tramo)
```

Este método se llamará tantas veces como tramos tengamos.

- Defina el método **GetTarifa** para calcular la tarifa según el número de minutos de un estacionamiento.
- Cree dos objetos de la clase **TarifadorParking** (uno para cada parking) y modifique adecuadamente el programa principal para calcular las tarifas a partir de los métodos de los objetos.

Mantenga la definición de la función **MinutosEntreInstantes** para calcular los minutos que hay entre dos instantes.

RELACIÓN DE PROBLEMAS IV. Vectores

Finalidad: Diseñar las cabeceras de los métodos que acceden a las componentes del vector. Dificultad Baja.

16. [Series ascendentes]. (*Examen Septiembre Doble Grado 2013*) Defina lo que sea necesario para calcular el número de series ascendentes de una secuencia de caracteres. Por ejemplo, la secuencia ttuvghtwwbde tiene 3 series ascendentes que son ttuv, ghtww, bde.

Para leer el mensaje cifrado debe leer caracteres con `cin.get()` (repase lo visto al inicio de esta relación de problemas) hasta que el usuario introduzca el carácter `#`. A continuación, el programa mostrará la cadena descodificada.

Ejemplo de entrada: ttuvghtwwbde# — Salida correcta: 3

Ejemplo de entrada: gfed# — Salida correcta: 4

Ejemplo de entrada: # — Salida correcta: 0

Finalidad: Recorrido sobre un vector procesando dos componentes en cada iteración. Dificultad Media.

17. [Elimina repetidos]. Sobre la clase `SecuenciaCaracteres`, añada un método `EliminaRepetidos` que quite los elementos repetidos, de forma que cada componente sólo aparezca una única vez. Se mantendrá la primera aparición, de izquierda a derecha. Por ejemplo, si la secuencia contiene
`{'b', 'a', 'a', 'h', 'a', 'a', 'a', 'c', 'a', 'a', 'a', 'g'}`
después de quitar los repetidos, se quedaría como sigue:
`{'b', 'a', 'h', 'c', 'g'}`

Implemente los siguientes algoritmos para resolver este problema:

- a) Usando un **vector local sin_repetidos** en el que almacenamos una única aparición de cada componente. Una vez construido, lo volcamos en `vector_privado`.

 Recorrer todas las componentes de "vector_privado"

 Si la componente NO está en "sin_repetidos",
 la añadimos

 Volcar "sin_repetidos" en "vector_privado"

- b) El problema del algoritmo anterior es que usa un vector local, lo que podría suponer una carga importante de memoria si trabajásemos con vectores grandes. Por lo tanto, vamos a resolver el problema sin usar vectores locales. Defina otra versión del método `EliminaRepetidos` que implemente el siguiente algoritmo:

 Recorrer todas las componentes de "vector_privado"

 Si NO es la primera aparición de la componente,
 Eliminarla

Para comprobar que no sea la primera aparición, basta buscarla a su izquierda (en la parte del vector privado que hay a su izquierda). Para eliminarla, basta llamar al método `Elimina` (de un único carácter).

- c) El anterior algoritmo nos obliga a desplazar muchas componentes cada vez que encontremos una repetida. Proponga una alternativa (sin usar vectores locales) para que el número de desplazamientos sea el menor posible e impleméntela.

Consejo: Use la misma técnica que se indicó en el ejercicio 14 de eliminar las ocurrencias de un carácter.

Construya un programa que lea los caracteres de la cadena uno a uno con `cin.get()`, hasta que se introduzca el carácter # y muestre el resultado de quitarle los repetidos. Puede probar la diferencia radical en el tiempo de ejecución de los algoritmos anteriores con el texto del Quijote, disponible en:

<http://decsai.ugr.es/~carlos/FP/Quijote.txt>

Mientras que la versión eficiente tarda algunos *milisegundos*, la versión ineficiente puede tardar *una hora*

Ejemplo de entrada: ggabaabghc# — Salida correcta: gabhc

Finalidad: Modificación de un vector a través de dos apuntadores. *Dificultad Media*.

18. [Elimina exceso de blancos]. Sobre la clase SecuenciaCaracteres, añada un método `EliminaExcesoBlancos` para eliminar el exceso de caracteres en blanco, es decir, que sustituya todas las secuencias de espacios en blanco por un sólo espacio. Por ejemplo, si la secuencia original es (' ', 'a', 'h', ' ', ' ', ' ', 'c'), que contiene una secuencia de tres espacios consecutivos, la secuencia resultante debe ser (' ', 'a', 'h', ' ', 'c').

Nota: Debe hacerse lo más eficiente posible.

Construya un programa que lea los caracteres de la cadena uno a uno con `cin.get()`, hasta que se introduzca el carácter # y muestre el resultado de quitarle el exceso de blancos. Puede probar el programa con el siguiente fichero, que contiene el Quijote con más de un espacio en blanco entre palabras:

http://decsai.ugr.es/~carlos/FP/Quijote_con_exceso_de_blanacos.txt

Finalidad: Recorrido de las componentes de un vector, en el que hay que recordar lo que ha pasado en la iteración anterior. *Dificultad Media*.

19. [Login]. (*Examen Febrero 2013*) Se está diseñando un sistema web que recopila datos personales de un usuario y, en un momento dado, debe sugerirle un nombre de usuario (login). Dicho login estará basado en el nombre y los apellidos; en concreto estará formado por los N primeros caracteres de cada nombre y apellido (en minúsculas, unidos y sin espacios en blanco). Por ejemplo, si el nombre es "Antonio Francisco Molina Ortega" y N=2, el nombre de usuario sugerido será "anfrmoor".

Debe tener en cuenta que el número de palabras que forman el nombre y los apellidos puede ser cualquiera. Además, si N es mayor que alguna de las palabras que aparecen en el nombre, se incluirá la palabra completa.

Por ejemplo, si el nombre es "Ana CAMPOS de la Blanca" y N=4, entonces la sugerencia será "anacampdelablan" (observe que se pueden utilizar varios espacios en blanco para separar palabras).

Implemente la clase **Login** y defina el método **Codifica** que recibirá una cadena de caracteres (tipo **string**) formada por el nombre y apellidos (separados por uno o más espacios en blanco) y devuelva otra cadena con la sugerencia de login.

```
class Login{  
private:  
    int num_caracteres_a_coger;  
public:  
    Login (int numero_caracteres_a_coger)  
        :num_caracteres_a_coger(numero_caracteres_a_coger)  
    { }  
    string Codifica(string nombre_completo){  
        .....  
    }  
};
```

Los únicos métodos que necesita usar de la clase **string** son **size** y **push_back**. Construya un programa que lea los caracteres de la cadena uno a uno con **cin.get()**, hasta que el usuario introduzca el carácter # y muestre el resultado de la codificación en pantalla.

Finalidad: Recorrido de las componentes de un vector, controlando qué ha ocurrido anteriormente. Dificultad Media.

20. [Cuenta Mayúsculas]. En este ejercicio no hay que definir ninguna clase. Todas las operaciones se realizan directamente en el **main**.

Construya un programa que vaya leyendo caracteres hasta que se encuentre un punto '.' y cuente el número de veces que aparece cada una de las letras mayúsculas. Imprimir el resultado.

Una posibilidad sería declarar un vector **contador_mayusculas** con tantas componentes como letras mayúsculas hay ('Z' - 'A' + 1) y conforme se va leyendo cada carácter, ejecutar lo siguiente:

```
cin >> letra;  
  
if (letra == 'A')
```

```
    contador_mayusculas[0] = contador_mayusculas[0] + 1;
else if (letra == 'B')
    contador_mayusculas[1] = contador_mayusculas[1] + 1;
else if (letra == 'C')
    contador_mayusculas[2] = contador_mayusculas[2] + 1;
else ....
```

Sin embargo, este código es muy redundante. Como solución se propone calcular de forma directa el índice entero que le corresponde a cada mayúscula, de forma que todos los **if-else** anteriores los podamos resumir en una **única** sentencia del tipo:

```
contador_mayusculas[indice] = contador_mayusculas[indice] + 1;
```

Hacedlo, declarando el vector directamente dentro del **main**.

Finalidad: Acceder a las componentes de un vector con unos índices que representen algo. Dificultad Baja.

21. **[Cuenta mayúsculas en una clase]**. Sobre el ejercicio 20, construya una clase específica **ContadorMayusculas** que implemente los métodos necesarios para llevar el contador de las mayúsculas. Lo que se pretende es que la clase proporcione los métodos siguientes:

```
void IncrementaConteo (char mayuscula)
int CuantasHay (char mayuscula)
```

El primer método aumentará en uno el contador de la correspondiente mayúscula y el segundo indicará cuántas hay. Modifique el programa principal para que cree un objeto de esta clase y llame a sus métodos para realizar los conteos de las mayúsculas. Finalmente, hay que imprimir en pantalla cuántas veces aparece cada mayúscula.

Finalidad: Diseño de una clase contadora de frecuencias. Dificultad Media.

22. **[Fibonacci]**. La sucesión de Fibonacci de orden n es una secuencia de números en la que los dos primeros son el 0 y el 1. A partir del tercero, los elementos se calculan como la suma de los n anteriores, si ya hay n elementos disponibles, o la suma de todos los anteriores si hay menos de n elementos disponibles.

Por ejemplo, la sucesión de Fibonacci de orden 4 sería la siguiente:

0, 1, 1, 2, 4, 8, 15, 29, ...

Defina una clase llamada **Fibonacci**. Para almacenar los enteros, se usará un vector de enteros. Al constructor se le pasará como parámetro el valor de n . Definid los siguientes métodos:

- **int GetBase()** para obtener el valor de n .

- void CalculaPrimeros(int tope) para que calcule los *tope* primeros elementos de la sucesión.
- int TotalCalculados() que devuelva cuántos elementos hay actualmente almacenados (el valor *tope* del método anterior)
- int k_esimo(int k) para que devuelva el elemento *k*-ésimo de la sucesión.

Escriba un programa que lea los valores de dos enteros, *n* y *k* y calcule, almacene y muestre por pantalla los *k* primeros términos de la sucesión de Fibonacci de orden *n*:

```
.....
Fibonacci fibonacci(n);

fibonacci.CalculaPrimeros(k);
tope = fibonacci.TotalCalculados(); // tope = k

for (int i=0; i<tope; i++)
    cout << fibonacci.k_esimo(i) << " ";
```

Dificultad Media.

23. [Eratóstenes]. (*Examen Septiembre 2012*) La **criba de Eratóstenes** (Cirene, 276 a. C. Alejandría, 194 a. C.) es un algoritmo que permite hallar todos los números primos menores que un número natural dado *n*.

El procedimiento consiste en escribir todos los números naturales comprendidos entre 2 y *n* y *tachar* los números que *no* son primos de la siguiente manera: el primero (el 2) se declara primo y se tachan todos sus múltiplos; se busca el siguiente número entero que no ha sido tachado, se declara primo y se procede a tachar todos sus múltiplos, y así sucesivamente. El proceso para cuando el cuadrado del número entero es mayor o igual que el valor de *n*.

El programa debe definir una clase llamada **Eratostenes** que contendrá:

- Como dato miembro debe declarar un vector privado **primos** tal que en la componente *k* se almacenará el primo *k*-ésimo ([2,3,5,7,...]). El cómputo de los primos se hará en el siguiente método.
- El método void **CalculaHasta(int n)** calcula los primos menores que *n*. Cuando se ejecute el método, se calcularán todos los primos menores que *n*, según el método de Eratóstenes descrito anteriormente.
Para realizar esta tarea, tendrá que definir un vector local al método con todos los números menores que *n* y procederá a *tachar* los no primos según el algoritmo de Eratostenes. Los números no tachados serán los primos y serán los que almacene en el dato miembro **primos**.

- El método `int TotalCalculados()` devuelva cuántos primos hay actualmente almacenados.
- `int k_esimo(int k)` para que devuelva el k -ésimo primo.

El programa principal quedaría de la forma:

```
Eratostenes primos;
int n = 100; int num_primos;

primos.CalculaHasta(n);
num_primos = primos.TotalCalculados();

for (int i=0; i<num_primos; i++)
    cout << primos.k_esimo(i) << " ";
```

Dificultad Media.

24. [Palabras en una frase]. (*Examen Septiembre Doble Grado 2013*) Defina una clase `Frase` para almacenar un conjunto de caracteres (similar a la clase `SecuenciaCaracteres`). Defina un método para localizar la k -ésima palabra.

Una palabra es toda secuencia de caracteres delimitada por espacios en blanco a izquierda y derecha. La primera palabra no tiene por qué tener espacios a su izquierda y la última no tiene por qué tener espacios a su derecha. Puede haber varios caracteres en blanco consecutivos.

Si k es mayor que el número de palabras, se considera que no existe tal palabra.

Por ejemplo, si la frase es `{' ', ' ', ' ', 'h', ' ', 'i', ' ', ' ', ' ', 'b', ' ', 'i', ' ', ' '}`. Si $k = 1$, la posición es 2. Si $k = 2$ la posición es 6. Si $k = 3$ la posición es -1.

Si la frase fuese `{'h', 'i', ' ', 'b', ' ', 'i', ' ', ' '}`, entonces si $k = 1$, la posición es 0. Si $k = 2$ la posición es 3. Si $k = 3$ la posición es -1.

Dificultad Media.

25. [Palabras en una frase -continuación-]. Sobre el ejercicio 24, añada los siguientes métodos:

- `void EliminaBlancosIniciales()` para borrar todos los blancos iniciales.
- `void EliminaBlancosFinales()` para borrar todos los blancos finales.
- `int NumeroPalabras()` que indique cuántas palabras hay en la frase.
- `void BorraPalabra(int k_esima)` para que borre la palabra k -ésima.
- `void MoverPalabraFinal(int k_esima)` para desplazar la palabra k -ésima al final de la frase.

Dificultad Media.

26. [Búsqueda por interpolación]. (*Examen Prácticas Septiembre 2016*) Implemente la **Búsqueda por Interpolación** en la clase `SecuenciaCaracteres`. El método busca un valor buscado entre las posiciones `izda` y `dcha` y recuerda a la *búsqueda binaria* porque requiere que el vector en el que se va a realizar la búsqueda esté ordenado y en cada consulta sin éxito se descarta una parte del vector para la siguiente búsqueda.

La diferencia fundamental con la búsqueda binaria es la manera en que se calcula el elemento del vector que sirve de referencia en cada consulta (que ocupa la posición `pos`). Ya no es el que ocupa la posición central del subvector en el que se efectúa la búsqueda (el delimitado únicamente por `izda` y `dcha`), sino que depende también del contenido de esas casillas, de manera que `pos` será más cercana a `dcha` si buscado es más cercano a `v[dcha]` y más cercana a `izda` si buscado es más cercano a `v[izda]`. En definitiva, se cumple la relación:

$$\frac{\text{pos} - \text{izda}}{\text{dcha} - \text{izda}} = \frac{\text{buscado} - v[\text{izda}]}{v[\text{dcha}] - v[\text{izda}]}$$

RELACIÓN DE PROBLEMAS V. Clases (Segunda parte)

1. [Elimina Varios]. Recupere el código de las clases SecuenciaCaracteres y SecuenciaEnteros disponible en:

<http://decsai.ugr.es/~carlos/FP/SecuenciaCaracteres.cpp>
<http://decsai.ugr.es/~carlos/FP/SecuenciaEnteros.cpp>

Dentro de la clase SecuenciaCaracteres defina un método EliminaVarios que elimine eficientemente todas las posiciones indicadas en una secuencia de enteros. Por ejemplo, si el vector de caracteres contiene Fundamentos, después de eliminar el conjunto de posiciones dado por 2 5 3, el vector se quedará como Fuaentos.

Observe que una posibilidad sería sustituir los caracteres a borrar por un carácter especial, por ejemplo # y luego pasarle un algoritmo que eliminase todas las ocurrencias de #. Sin embargo, debemos evitar esta técnica ya que no podemos presuponer que tenemos la posibilidad de elegir tal carácter especial en cualquier secuencia.

Se recomienda implementar el siguiente algoritmo:

Utilizar dos índices: pos_escritura y pos_lectura que marquen las posiciones de lectura y escritura en el vector de char

Ordenar la secuencia de enteros

Recorrer con pos_lectura los caracteres del vector de char
Si el carácter actual no está en una posición a borrar,
colocarlo en pos_escritura.

Construya un programa principal que lea los caracteres de la secuencia de caracteres hasta encontrar el terminador #, luego lea la secuencia de posiciones a eliminar (secuencia de enteros positivos hasta encontrar el -1) e imprima el resultado de eliminar dichas posiciones.

Observe que en este programa principal se ha elegido el terminador # pero en otro programa podría usarse otro terminador distinto. Es por ello que no podemos implementar el método EliminaVarios usando la primera técnica indicada al inicio del enunciado de este ejercicio, ya que el carácter # podría ser un carácter válido en otro programa.

Ejemplo de entrada: abcdefghij#9 0 2 1 5 -1 -- Salida correcta: deghi

Finalidad: Métodos que reciben objetos como parámetros. Dificultad Media.

2. [Circunferencia]. Recupere la solución del ejercicio 48 de la relación de problemas III (Circunferencia representando el centro con un struct) Sobre la clase

Circunferencia, defina el método `SegmentoRadio` para que devuelva un objeto de la clase `SegmentoDirigido` con el segmento correspondiente al radio de la circunferencia.

Cree un programa principal que lea el centro (dos datos de tipo `double`) y la longitud del radio (un dato de tipo `double`), construya una circunferencia con estos datos y a partir de ella, construya el segmento correspondiente al radio. Calcule ahora la longitud del segmento radio a través del método `Longitud` de la clase `SegmentoDirigido` y muestre el resultado en pantalla (debe coincidir con la longitud original del radio)

Finalidad: Métodos que devuelven objetos de otra clase. Dificultad Baja.

3. [Lector tarifador parking]. Recupere la solución del ejercicio 46 de la relación de problemas III (Parking con clases y struct) Construya la clase `LectorTarifador` cuyo objetivo es realizar las entradas de datos para poder construir objetos de la clase `TarifadorParking`. En concreto, debe proporcionar los siguientes métodos:

```
class LectorTarifador{  
    ....  
    TarifadorParking LeeTarifador_TarifaDiaCompletoUltimoDato()  
    ....  
    TarifadorParking LeeTarifador_TarifaDiaCompletoPrimerDato()  
    ....  
};
```

El primer método leerá desde el dispositivo de entrada de datos por defecto los datos tal y como se especificó en el ejercicio 46 de la relación de problemas III, es decir, el número de tramos, las parejas formadas por los límites y las tarifas de cada tramo y finalmente la tarifa de día completo. Con dichos datos, el método construirá un objeto de la clase `TarifadorParking` y lo devolverá.

El segundo método realizará una lectura parecida, salvo que la lectura de la tarifa de día completo se hará antes que la lectura del número de tramos.

Reescriba el programa principal para que cree los objetos `tarifador_parking_1` y `tarifador_parking_2` a través de un objeto de la clase `LectorTarifador`.

Observe que el segundo método (`LeeTarifador_TarifaDiaCompletoPrimerDato`), aunque hay que implementarlo, no lo usaremos en este ejercicio, ya que en el programa principal pedido, se indica que la tarifa de día completo se lee después de los tramos.

Finalidad: Métodos que devuelven objetos de otra clase. Dificultad Media.

4. [Descodifica en la clase SecuenciaCaracteres]. (Examen Febrero 2016) Recupere la solución del ejercicio 5 (Descodifica) de la relación de problemas IV. Implemente el método de descodificación como un método de la clase `SecuenciaCaracteres`.

El método no modificará la secuencia de caracteres sobre la que se aplica, sino que construirá y devolverá una nueva secuencia.

Finalidad: Métodos que devuelven objetos de la misma clase. Dificultad Baja.

5. [Moda]. Recupere la solución del ejercicio 10 de la relación de problemas IV (Moda). Re-escriba el método Moda de la siguiente forma:

- Use como dato local para guardar los caracteres procesados un objeto de la clase SecuenciaCaracteres, en vez de un vector clásico de corchetes.
- Defina dentro de la clase SecuenciaCaracteres el método

```
int NumeroOcurrencias(char buscado, int izda, int dcha)
```

que devuelve el número de ocurrencias de un valor buscado entre las posiciones izda y dcha de la secuencia. Llame a este método dentro de Moda para calcular las frecuencias de cada carácter.

Finalidad: Métodos con objetos de la misma clase como datos locales. Dificultad Baja.

6. [Moda vs 2]. Recupere la solución del ejercicio 5 (Moda) Elimine el método NumeroOcurrencias y en su lugar defina el método

```
SecuenciaEnteros PosicionesOcurrencias  
(char buscado, int izda, int dcha)
```

que devuelve las posiciones en las que se encuentra el elemento buscado. Para ello, tendrá que usar la clase SecuenciaEnteros, análoga a SecuenciaCaracteres, cuya definición puede encontrarse en:

<http://decsai.ugr.es/~carlos/FP/SecuenciaEnteros.cpp>

Cambie la implementación del método Moda del ejercicio 5 para que use este nuevo método

Finalidad: Métodos que devuelven objetos de otra clase. Dificultad Baja.

7. [Elimina repetidos con una secuencia local]. En el primer apartado del ejercicio 17 de la relación de problemas IV se pedía eliminar los repetidos de un objeto de la clase SecuenciaCaracteres usando como dato auxiliar local un vector clásico con los elementos que no estuviesen repetidos. Recupere la solución de dicho ejercicio y resuélvalo usando como dato auxiliar local un objeto de la propia clase SecuenciaCaracteres

Finalidad: Métodos con objetos de la misma clase como datos locales. Dificultad Baja.

8. [Inserta secuencia ineficiente]. Sobre la clase SecuenciaCaracteres, defina un método que inserte otra secuencia a partir de una posición.

Implemente este método utilizando el método visto en clase que insertaba un único carácter:

```
void Inserta(int pos_insercion, char valor_nuevo)
```

Tendrá que llamar a este método para insertar, uno a uno, todos los caracteres de la nueva secuencia.

Construya un programa para que lea los caracteres de la primera secuencia (hasta el terminador #), los caracteres de la segunda (hasta el terminador #), la posición de inserción e imprima el resultado de insertar la segunda dentro de la primera, a partir de la posición indicada.

Observe que este algoritmo es bastante ineficiente ya que el algoritmo se reduce a la ejecución de un bucle anidado en otro. En el ejercicio 9 se pide una versión eficiente.

Ejemplo de entrada: `Esto es fácil#muy #8`

— Salida correcta: `Esto es muy fácil`

Finalidad: Métodos que reciben como parámetro un objeto de la misma clase. Dificultad Baja.

9. [Inserta secuencia eficiente]. Implemente el método del ejercicio 8 de una forma eficiente con un único bucle (sin tener que llamar al método de inserción de un carácter)

Finalidad: Métodos que reciben como parámetro un objeto de la misma clase. Dificultad Media.

10. [Parking con la clase Instante]. Recupere la solución del ejercicio 46 de la relación de problemas III (Parking con clases y struct)

Re-escríbalo para que los instantes sean objetos de la clase Instante. Por lo tanto, tiene que eliminar la función que calculaba los minutos entre dos instantes y hacerlo dentro de la clase Instante.

Finalidad: Métodos que reciben como parámetro un objeto de la misma clase. Dificultad Baja.

11. [k Mayores que, dentro de una clase]. ([Examen Septiembre 2016](#)) Utilice la definición de la clase SecuenciaDoubles disponible en

<http://decsai.ugr.es/~carlos/FP/SecuenciaDoubles.cpp>

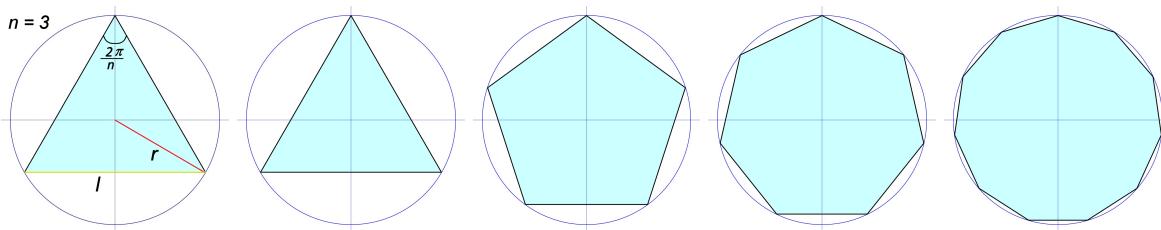
Esta clase es análoga a SecuenciaCaracteres, pero trabaja con datos de tipo double.

Defina un método que construya otro objeto de la misma clase SecuenciaDoubles que contenga los k elementos ordenados mayores que un valor de referencia, según lo indicado en el ejercicio 3 de la relación de problemas IV. Modifique el programa principal de dicho ejercicio para que utilice la clase SecuenciaDoubles.

Finalidad: Métodos que devuelven un objeto de la misma clase. Dificultad Media.

12. [Polígono]. ([Examen Febrero 2016](#)) Un polígono regular de n caras tiene n lados de la misma longitud y todos los ángulos interiores son iguales. Su centro geométrico es

el centro de la circunferencia circunscrita (la que lo envuelve). Supondremos que dos polígonos son distintos si se diferencian o bien en sus centros geométricos, o bien en el número de lados o bien en la longitud de cualquiera de ellos. Así pues, por ejemplo, no tendremos en cuenta las distintas posiciones en el plano que se podrían obtener girando el polígono sobre su centro.



Si llamamos n al número de lados y l la longitud de cualquiera de ellos, tenemos que:

- la longitud r del radio de la circunferencia circunscrita viene definida por $r = l/(2 \sin(\pi/n))$
- El área del polígono es $A = (n/2) \cdot r^2 \sin(2\pi/n)$
- Si queremos construir un polígono inscrito en la misma circunferencia, pero multiplicando por un entero k el número de lados, la longitud de cada uno de los kn lados viene dada por $l' = r\sqrt{2(1 - \cos(2\pi/(kn)))}$.

Se quiere diseñar la clase `PolygonoRegular` para poder representar este tipo de polígonos y realizar las siguientes tareas:

- Calcular el perímetro del polígono.
- Calcular el área del polígono.
- Calcular la diferencia entre el área del polígono y la del círculo correspondiente a la circunferencia circunscrita.
- Comprobar si un polígono es más grande que otro (considerando al área de cada uno)
- Construir un nuevo polígono que tenga la misma circunferencia circunscrita y con un número de lados que sea múltiplo del número de lados del polígono.

Debe tener los siguientes constructores:

- Un constructor sin parámetros en el que los valores a asignar por defecto sean: 3 para el número de lados (triángulo), 1 para la longitud y (0,0) para las coordenadas del centro.
- Un constructor que cree un polígono regular con una longitud y número de lados concretos y deje como centro el valor por defecto (0,0).

- Un constructor que cree un polígono regular con una longitud, número de lados y centro concretos.

Defina un programa que realice lo siguiente:

- Cree dos polígonos, `poligono1` con los valores por defecto y `poligono2` con 6 lados de longitud 4 y centrado en (0,0). El programa comprobará si `poligono1` es más grande que `poligono2`.
- Construya un nuevo polígono a partir de `poligono1`, con la misma circunferencia circunscrita y con el doble número de lados. El programa imprimirá en pantalla el área del nuevo polígono.
- Repita el proceso anterior generando polígonos con el doble número de lados en cada iteración, hasta que el polígono generado tenga un área *similar* a la del círculo delimitado por la circunferencia circunscrita. Consideraremos que las áreas son similares si no se diferencian en más de 10^{-5} .

El programa mostrará el número de lados del polígono que aproxima a la circunferencia (el resultado con `poligono1` es 1536 lados)

Finalidad: Trabajar con constructores y con métodos que devuelven y reciben objetos.
Dificultad Media.

13. [Counting Sort]. (*Examen Septiembre 2014*) Sobre la clase SecuenciaCaracteres implemente el algoritmo **Counting Sort** para ordenar sus valores.

`SecuenciaCaracteres CountingSort()`

El método no modificará las componentes del vector privado sino que debe construir una secuencia nueva y devolverla. El algoritmo funciona de la siguiente forma:

- Calcular los valores mínimo y máximo del vector. Por ejemplo, si el vector contiene
c b b a b c c a g c b g c
el mínimo es 'a' y el máximo 'g'.
- Construir un vector local de frecuencias con los conteos de todos los caracteres que hay entre el mínimo y el máximo. Con el ejemplo anterior, el vector de conteos será

2 4 5 0 0 0 2

que corresponden a las frecuencias de las letras que hay entre 'a' y 'g'.

- Recorrer el vector de frecuencias almacenando cada carácter tantas veces como indique su frecuencia (2 veces el 'a', cuatro veces el 'b', etc)

a a b b b c c c c g g

Para resolver este problema, debe definir un segundo método auxiliar:

SecuenciaCaracteres CountingSortEntre(char min, char max)

Este método sólo ordena los valores de la secuencia que hay entre un carácter **izquierda** y otro carácter **derecha**. Por ejemplo, si **izquierda** = 'b' y **derecha** = 'g' el resultado sería la siguiente secuencia:

b b b b c c c c g g

Debe llamar a este método dentro de **CountingSort()**.

Finalidad: Trabajar con métodos con vectores locales y devolviendo un objeto de la misma clase. Dificultad Media.

14. [Palabra desordenada]. (*Examen Examen Febrero 2009*)

Sgeún un etsduio de una uivenrsdiad ignlsea, no ipmotra el odren en el que las ltears etsan ersciats, la úicna csoa ipormtnate es que la pmrireia y la última ltera etsén ecsritas en la psioción cocrrtea. El rsteo peuden estar ttaolmntee mal y aún pordás lerelo sin pobrleams. Etso es pquore no lemeos cada ltera por sí msima snio la paalbra cmoo un tdo.

Diremos que dos palabras son similares si la primera letra de ambas palabras es igual, la última letra de ambas palabras también es igual y el resto de las letras son las mismas pero no están necesariamente en las mismas posiciones. De esta forma, las palabras ttalomntee y totalmente son similares.

Sobre la clase **SecuenciaCaracteres**, defina un método que compruebe si la secuencia es similar a otra.

Construya un programa principal que lea los caracteres de la primera secuencia hasta el terminador #, lo mismo con la segunda y nos diga si son similares.

Finalidad: Trabajar con métodos a los que se les pasa como parámetros objetos de la misma clase. Dificultad Media.

15. [Circunferencia con centro un Punto2D]. Recupere la solución del ejercicio 2 de esta relación de problemas (Circunferencia)

Cámbielo para representar el centro de la circunferencia con un objeto de la clase **Punto2D**.

Finalidad: Trabajar con objetos como datos miembros de otros objetos. Dificultad Media.

16. [Circunferencia circunscrita]. Recupere las implementaciones de las clases **Punto2D**, **SegmentoDirigido**, **Circunferencia** y **Cuadrado**. Las tres primeras se han visto en clase de teoría. Con respecto al cuadrado, éste viene determinado por el punto correspondiente a la esquina inferior izquierda y por la longitud de cualquiera de sus lados (estos serán sus datos miembros). Supondremos que sólo representamos cuadrados cuya base es paralela al eje de las abscisas.

```
class Punto2D{
private:
    double abscisa;
    double ordenada;
public:
    .....
};

class SegmentoDirigido{
private:
    double x_1, y_1, x_2, y_2;
public:
    .....
};

class Circunferencia{
private:
    double centro_x;
    double centro_y;
    double radio;
public:
    .....
};

class Cuadrado{
private:
    double esquina_abscisa;
    double esquina_ordenada;
    double longitud;
public:
    .....
};
```

Definid sobre la clase Cuadrado los siguientes métodos:

- Métodos para calcular el área y el perímetro del cuadrado.
- Obtener el punto central interior al cuadrado:

Punto2D Centro()

Para calcular las coordenadas basta sumar la mitad de la longitud del cuadrado a las coordenadas de la esquina inferior izquierda.

- Obtener la circunferencia inscrita al cuadrado (la que está por dentro):

Circunferencia CircunferenciaInscrita()

Esta circunferencia es la que tiene como centro el centro del cuadrado y como radio la mitad de la longitud del cuadrado.

- Obtener la circunferencia circunscrita al cuadrado (la que está por fuera):

Circunferencia CircunferenciaCircunscrita()

Esta circunferencia es la que tiene como centro el centro del cuadrado y como radio, la longitud del segmento que une el centro con la esquina inferior izquierda. Obtened la longitud creando el objeto de la clase SegmentoDirigido y a continuación llamad al método Longitud.

- Determinar si un cuadrado tiene mayor área que otro.

Complete el programa principal de prueba que se encuentra en el fichero **FigurasGeometricas.cpp**

Finalidad: Trabajar con el constructor de copia y con métodos que devuelven objetos. Dificultad Baja.

17. [Cuadrado dentro de otro]. (*Examen Febrero 2012*) Sobre el ejercicio anterior, implemente un método para determinar si un cuadrado contiene a otro. Un cuadrado C_1 determinado por la esquina (x_1, y_1) y la longitud l_1 contiene a otro cuadrado C_2 dado por (x_2, y_2) y l_2 si se cumple que $x_2 \geq x_1$ y $x_2 + l_2 \leq x_1 + l_1$ y $y_2 \geq y_1$ y $y_2 + l_2 \leq y_1 + l_1$

Finalidad: Pasar a un método de una clase un parámetro de la misma clase. Dificultad Baja.

18. [Replace]. (*Examen Septiembre 2014*) Existe un método para la clase **string** de C++, denominado **replace**, que cambia **n** caracteres de una cadena **cad1**, empezando en una determinada posición **pos**, por los caracteres presentes en una segunda cadena **cad2**. La llamada al método es **cad1.replace(pos, n, cad2)**. Ejemplos del funcionamiento de **replace** son:

```
string cad1="Fundamental Programación";
cad1.replace(9,2,"os de la");    // "al" -> "os de la"
                                // Ahora cad1 tiene "Fundamentos de la Programación"
cad1.replace(12,5,"en");        // "de la" -> "en"
                                // Ahora cad1 tiene "Fundamentos en Programación"
```

Puede observar que, dependiendo de la cadena a insertar y de las posiciones especificadas, la secuencia final puede ser más grande o más pequeña que la original.

Defina sobre la clase **SecuenciaCaracteres** el método **Replace** para que haga la tarea pedida. Tendrá que pasarle al método la posición inicial, el número de caracteres a eliminar y el objeto de la clase **SecuenciaCaracteres** conteniendo la secuencia de caracteres de reemplazo.

Construya un programa de prueba que lea caracteres hasta el terminador # y los almacene en una primera secuencia. Haga lo mismo para una segunda secuencia. A continuación, el programa leerá dos enteros **pos** y **n** y procederá a reemplazar los **n**

RELACIÓN DE PROBLEMAS V. Clases (Segunda parte)

caracteres que hay a partir de la posición `pos` del primer vector, con los caracteres del segundo vector.

Restricciones para este ejercicio: No se puede utilizar la clase `string` en ninguna parte del programa, debe hacerse lo más eficiente posible y no puede utilizarse una tercera secuencia o vector en el que se vaya almacenando el resultado, es decir, las modificaciones deben hacerse directamente sobre la primera secuencia.

Finalidad: Trabajar con métodos a los que se les pasa como parámetro y definen objetos locales de la MISMA clase.

Dificultad Media.

19. [Conjunto ordenado]. (*Examen Septiembre 2012*) Defina la clase `ConjuntoOrdenado` para que permita almacenar una secuencia **ordenada** de números enteros **sin repetidos**. Defina métodos para:

- Añadir un entero (de forma ordenada y sin almacenar repetidos).
- Calcular la unión con otro conjunto. En la unión se deben incluir los elementos que estén en cualquiera de ellos.
- Calcular la intersección con otro conjunto. En la intersección se deben incluir los elementos que sean comunes a ambos conjuntos.

Cree un programa principal de prueba.

Finalidad: Trabajar con métodos a los que se les pasa como parámetro y devuelven objetos de la misma clase. Dificultad Media.

20. [Secuencia cíclica]. (*Examen Septiembre 2009*) Sobre la clase `SecuenciaCaracteres`, añada un método que determine si dicha secuencia de caracteres C_1 contiene a otra secuencia C_2 en el mismo orden (no tienen que estar consecutivos) y de forma *cíclica*. Para que se cumpla este criterio se deben satisfacer las siguientes condiciones

- Todos los caracteres de C_2 deben estar en C_1
- Deben estar en el mismo orden aunque no de forma consecutiva
- Si durante la búsqueda se ha llegado al final de la secuencia C_1 , se debe proseguir la búsqueda por el inicio de C_1 , pero sin sobrepasar la posición en la que hubo la primera concordancia.

Se muestran algunos ejemplos en los que la secuencia C_1 contiene a C_2 :

- $C_1 = xz\underline{ayobnmcpwqdfg}$ $C_2 = abcd$
- $C_1 = ftk\underline{cpxqdhjzxqoblki}$ $C_2 = abcd$
- $C_1 = tzs\underline{bluyclpygdmngrafvc}$ $C_2 = abcd$

RELACIÓN DE PROBLEMAS V. Clases (Segunda parte)

Hay que destacar que la primera letra de C_2 a buscar en C_1 podría estar en cualquier sitio. Por ejemplo, para el siguiente caso:

- $C_1 = \underline{bghcjadxak}$ $C_2 = abcd$

podemos ver que a partir de la primera \boxed{a} de C_1 no podemos encontrar C_2 de forma cíclica, aunque sí lo podemos hacer a partir de la segunda \boxed{a} de C_1 .

También puede darse el caso de que C_1 no contenga a C_2 de forma cíclica aunque incluya todas sus letras, como muestra el siguiente ejemplo:

- $C_1 = tzsbluyclpcaygdmxngrfvc$ $C_2 = abcd$