



Fundamentos de Programación.

Guión de Prácticas.

Curso 2014/2015

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 guión de prácticas

El guión está dividido en sesiones. En cada sesión se plantean una serie de problemas de programación a resolver. Cada semana se entregará una sesión con los 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 se hará la semana siguiente 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 guión. Dichas actividades no se entregarán al profesor. Terminada la defensa, el profesor explicará los ejercicios a todos los alumnos. 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 son de varios tipos:

- **Básicos:** Son los problemas que, obligatoriamente, el alumno debe resolver en cada sesión. Del conjunto de problemas básicos, en el guión de prácticas se distinguirá entre:
 1. **Obligatorios:**
Si se realizan correctamente estos ejercicios, el alumno podrá sacar hasta un 9 (sobre 10) en la nota de prácticas.
 2. **Opcionales:**
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 ejercicios opcionales.
- **Ampliación:** problemas cuya solución no se verá, pero que sirven para afianzar conocimientos. El alumno debería intentar resolver por su cuenta un alto porcentaje de éstos.

Para la realización de estas prácticas, se utilizará el entorno de programación Orwell Dev C++. En la página 4 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 guiones.

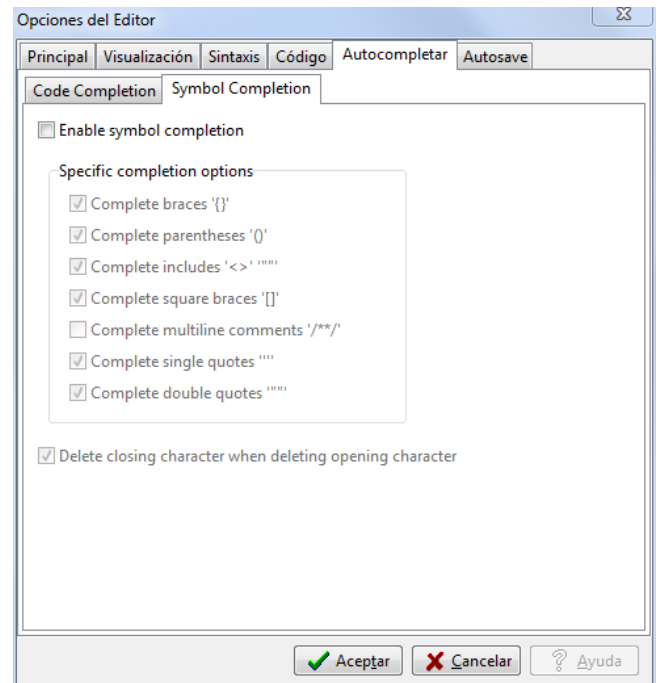
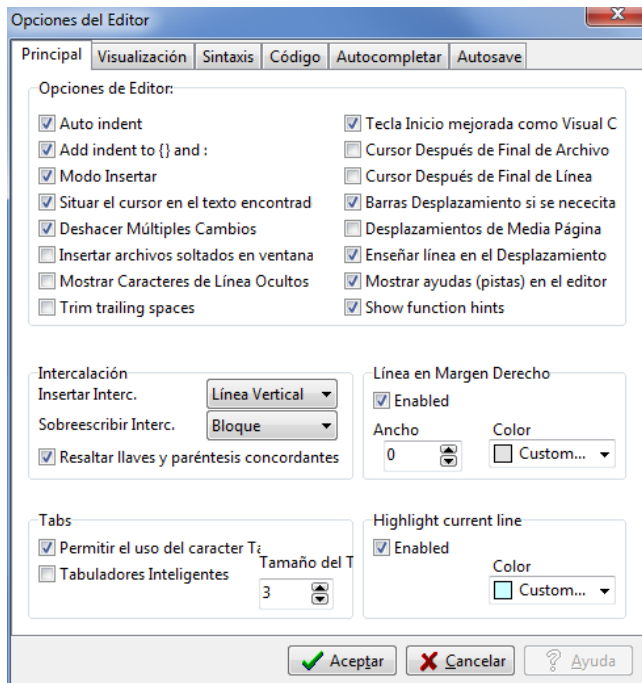
Instalación de Orwell Dev C++ en nuestra casa

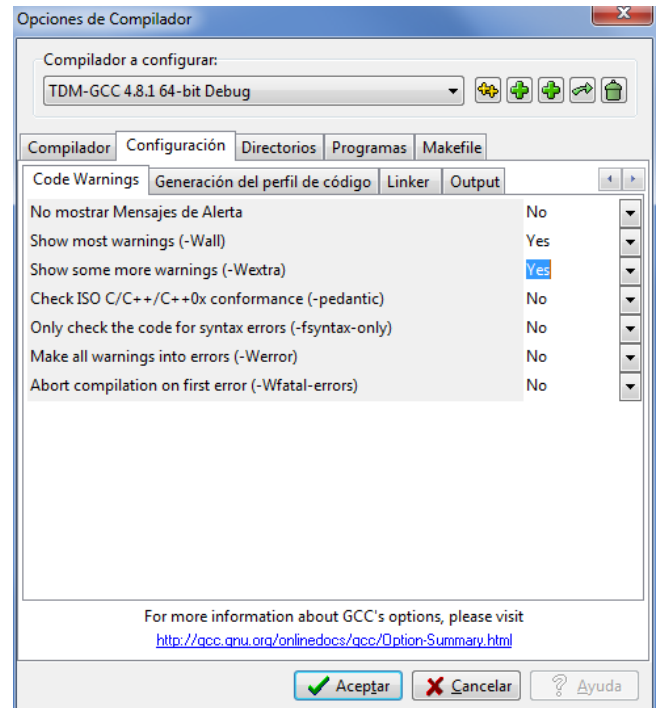
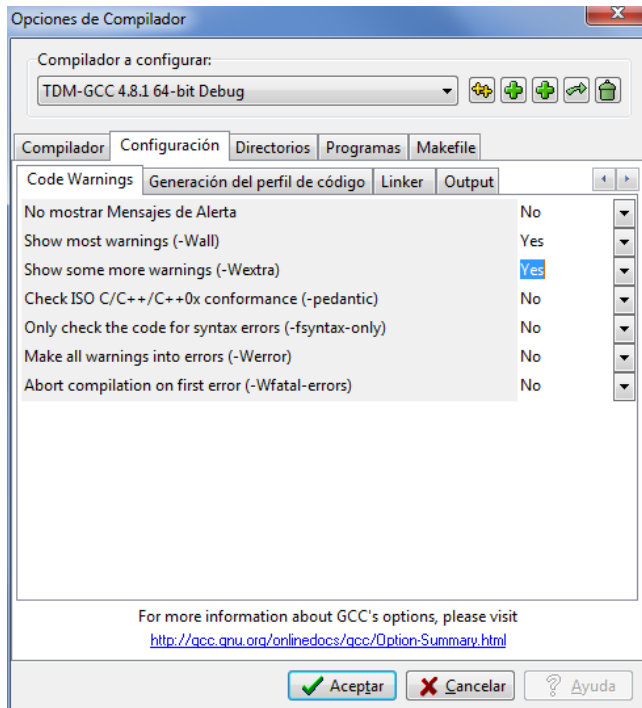
El entorno de desarrollo que usaremos será Orwell Dev C++. Puede descargarse desde la página:

http://sourceforge.net/projects/orwelldvcpp/?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
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 dos alternativas:

a) 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 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.

b) Si queremos hacerlo para una única consola, basta ejecutar el comando

`chcp 1252`

sobre la consola. El problema es que cada vez que se abre una nueva consola (por ejemplo, como resultado de ejecutar un programa desde Orwell Dev C++) hay que realizar este cambio. En nuestro caso, pondríamos (por ejemplo, al inicio del programa, justo después de las declaraciones de las variables) lo siguiente:

`system("chcp 1252");`

En cualquier caso, remarcamos que esta solución no es necesaria si se adopta la primera, es decir, el cambio del registro de Windows.

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

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

La primera sesión de prácticas tendrá lugar la segunda semana de clase. En cualquier caso, antes de esta primera sesión, el alumno debe realizar las siguientes tareas en su casa:

► **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 en 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++. Consultad la sección de Instalación (página 4) de este guión.

Actividad: Resolución de problemas.

Realizad una lectura rápida de las actividades a realizar durante la próxima sesión de prácticas en las aulas de ordenadores (ver página siguiente) e intentad resolver en papel los ejercicios que aparecen en la página 16

Actividades de Ampliación

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

<http://loro.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 la primera sesión de prácticas (que tendrá lugar la segunda semana de clase) Esta es la única sesión en la que el alumno no tendrá que entregar nada previamente.

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. Si durante la sesión se requiere abrir algún fichero, éste puede encontrarse en la plataforma web de la asignatura <https://decsai.ugr.es> (acceso identificado) o en la carpeta del Sistema Operativo instalado en las aulas

H:\CCIA\Grado_FP\

En el escritorio de Windows, se encuentra un acceso directo a dicha carpeta.

Muy Importante. Los ficheros que se encuentran en la unidad H: están protegidos y no pueden modificarse. Por tanto, habrá que copiarlos a la unidad local U:, dónde ya sí podrán ser modificados.

El primer programa

Copiando el código fuente

En el directorio H:\ccia\Grado_FP\ProblemasI se encuentra el directorio I_Pitagoras. Copiadlo entero a vuestra carpeta local (dentro de U:\FP).

Importante: Siempre hay que copiar localmente las carpetas que aparecen en la unidad H: del departamento ya que están protegidos contra escritura y no se puede trabajar directamente sobre ellos.

Desde el Explorador de Windows, entrad en la carpeta recién creada en vuestra cuenta:

U:\FP\I_Pitagoras

y haces doble click sobre el fichero I_Pitagoras.cpp. Debe aparecer una ventana como la de la figura 1

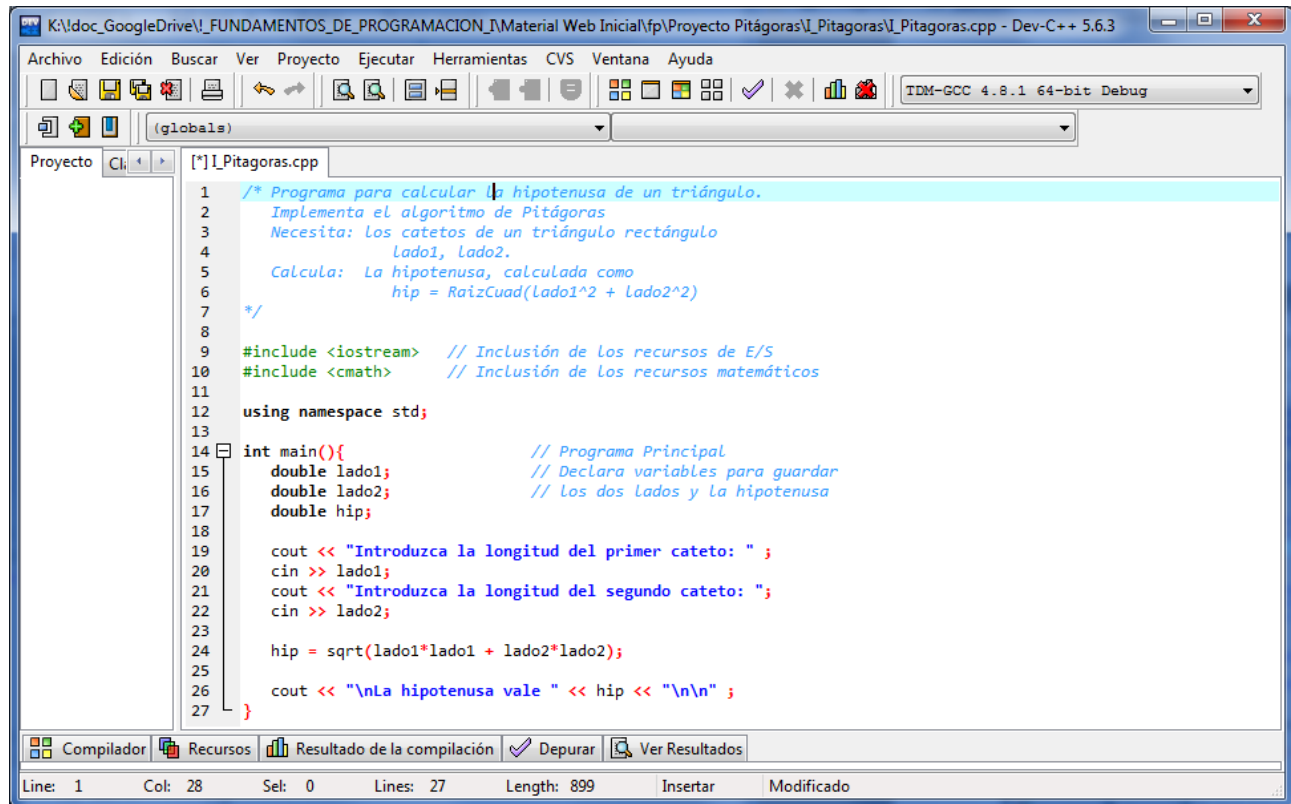


Figura 1: Programa que implementa el algoritmo de Pitágoras

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.

The image shows a C++ code snippet with several handwritten annotations in blue and red ink. On the left, blue annotations group lines of code: 'Declaraciones' for the variable declarations, 'Entradas datos' for the input statements, 'Computos' for the calculation, and 'Salida Resultados' for the output and pause statements. Red annotations include 'líneas en blanco' with arrows pointing to empty lines, 'espacios en blanco' with arrows pointing to spaces around operators and assignment, and a red circle around the first three lines of code with the text 'Comentarios separados visualmente del código'.


```
int main() {  
    double lado1;  
    double lado2;  
    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" ;  
    system("pause");  
}
```

Figura 2: Escritura de código

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



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.

Introducíd 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`.
- Quitad 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 `dpuble`
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 *avisos*. 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*lado2)`
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 1 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 (Ctrl-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

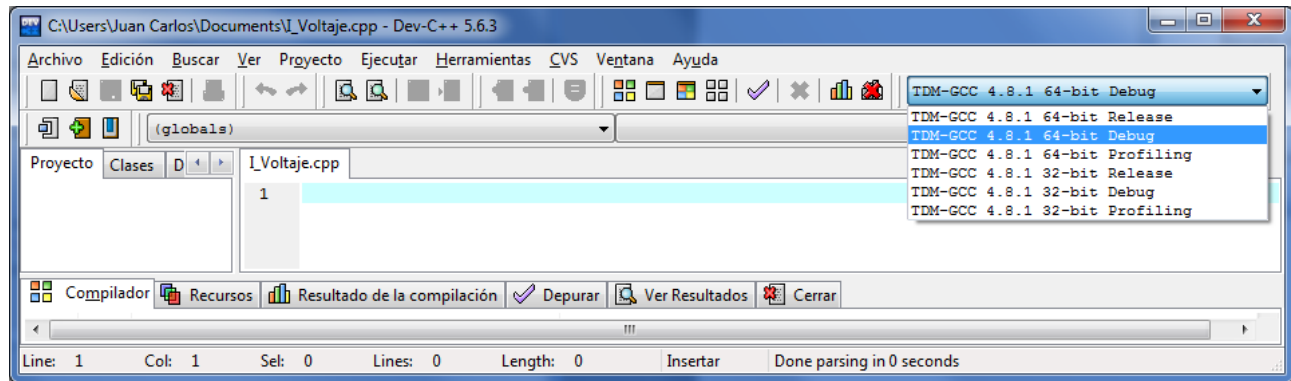


Figura 3: Creación de un programa nuevo

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.

Resolved los ejercicios siguientes de la relación de problemas I, página **RP-I.1**.

- 2 (Interés bancario)
- 3 (Circunferencia)
- 4 (Gaussiana)

Sesión 2

Tipos básicos y operadores

► **Actividades a realizar en casa**

Actividad: Resolución de problemas.

Resolved los siguientes problemas de la relación I. Recordad que antes del inicio de esta sesión en el aula de ordenadores hay que subir las soluciones a `decsai.ugr.es`, tal y como se explica en la página 2. Debe subir un fichero llamado `sesion2.zip` que incluya todos los ficheros `cpp`.

- *Obligatorios:*
 - 5 (Fabricante)
 - 6 (Intercambiar dos variable)
 - 7 (Media aritmética y desviación típica)
 - 8 (Pinta dígitos)

Actividades de Ampliación

Recordad los conceptos de combinación y permutación, que irán apareciendo recurrentemente a lo largo de la carrera. Consultad, por ejemplo

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

para una introducción básica a los conceptos.

Ejecutad cualquiera de los siguientes applets

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

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

con los valores pertinentes para calcular cuántos número distintos se pueden representar utilizando únicamente dos símbolos (0 y 1) en 64 posiciones de memoria. Ejecutad el mismo applet para que muestre en la parte inferior las distintas permutaciones que se pueden conseguir con 2 símbolos (0,1) y 4 posiciones.



► **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 siguientes de la Relación de Problemas I, página **RP-I.1**. Estos ejercicios no han de entregarse en decsa1.

19 (PVP automóvil)

22 (Funciones matemáticas)

Sesión 3

Estructura condicional

► **Actividades a realizar en casa**

Resolved los siguientes problemas:

- *Obligatorios:*

De la relación de Problemas I:

- 9 (Horas, minutos, segundos)
- 11 (Intercambiar tres variables)
- 14 (Pasar de mayúscula a minúscula)
- 15 (Pasar de carácter a entero)
- 17 (Expresiones lógicas)

De la relación de Problemas II:

- 1 (Valor por encima o por debajo de la media)
- 3 (Ver si dos números se dividen)

- *Opcionales:*

- 10 (Precisión y desbordamiento)

Actividades de Ampliación

Hojea 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

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:

- Suprimid los paréntesis de alguna de las expresiones lógicas de la sentencia `if`
- Quitad 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)
- Quitad 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.

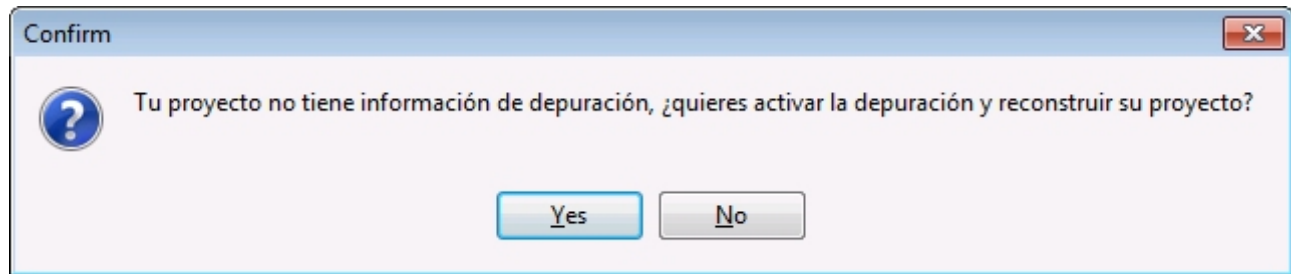


Figura 4: Ventana emergente que aparece cuando la configuración actual del compilador no permite tareas de depuración

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 línea que contiene la primera sentencia condicional `if` (figura 5).

2. Comenzar la depuración:

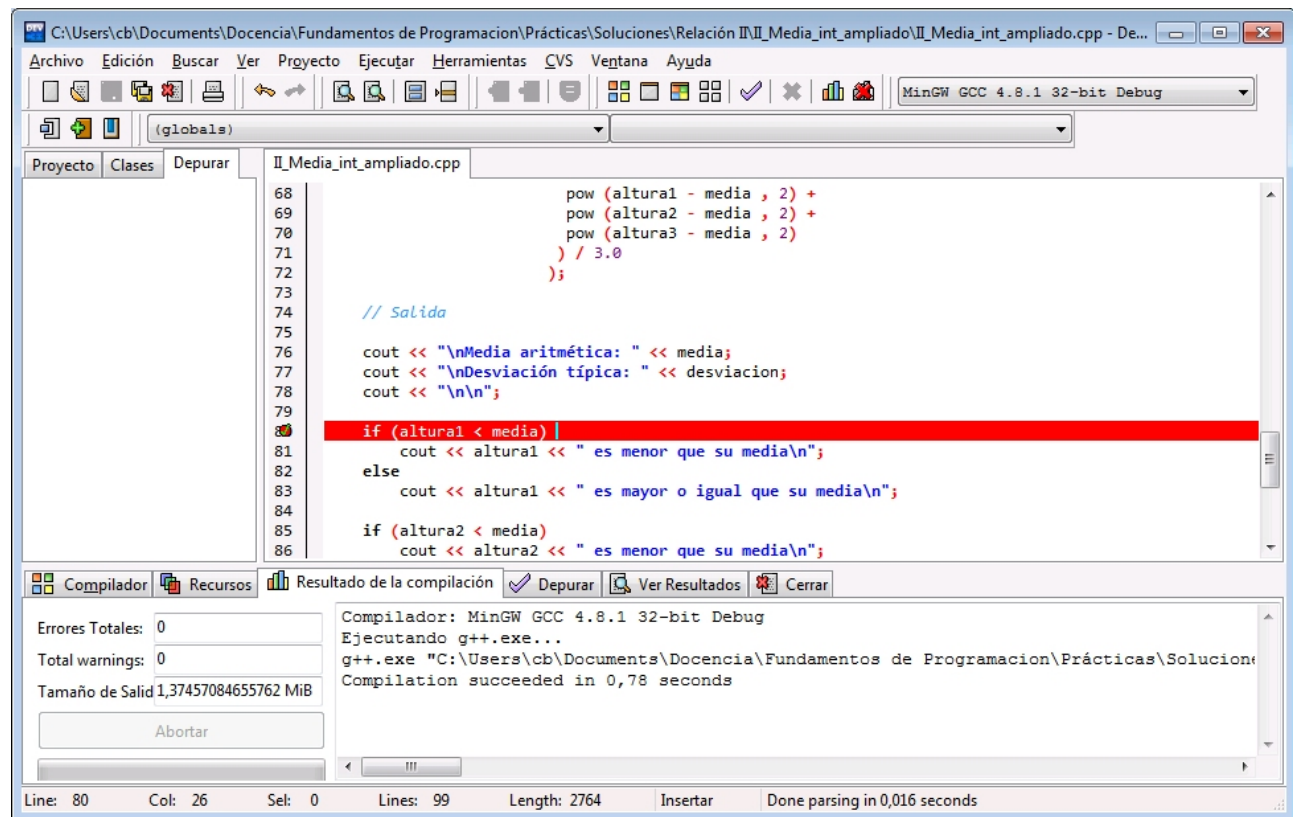



Figura 5: Se ha activado un punto de ruptura

- 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)

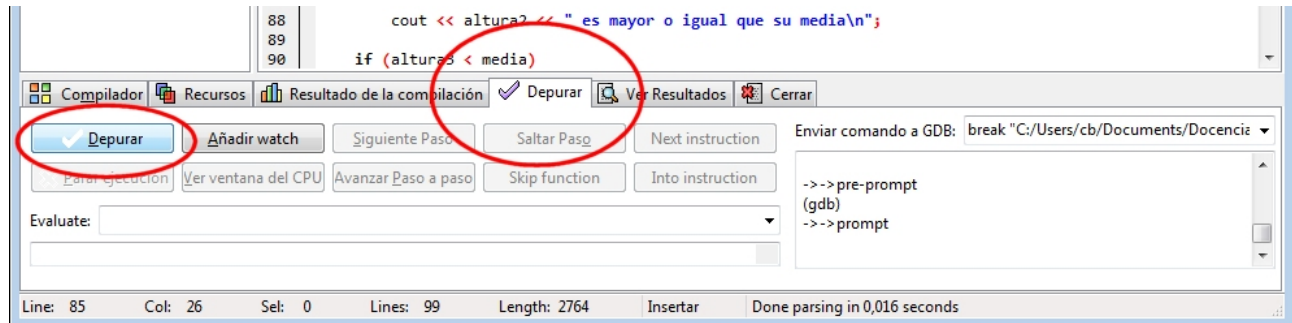


Figura 6: Inicio del proceso de depuración

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.

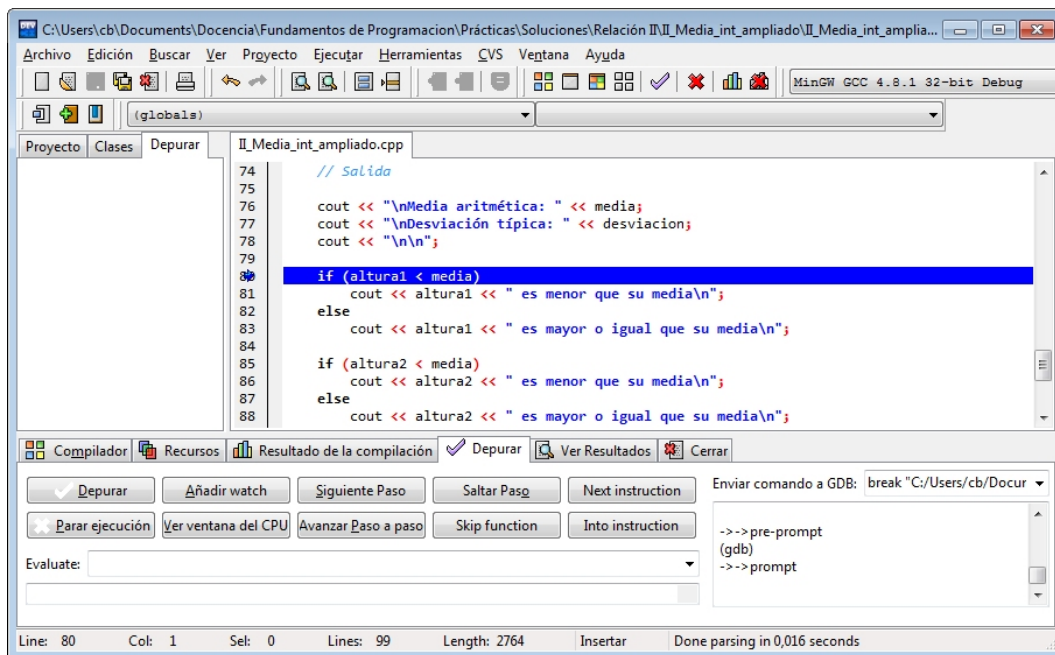


Figura 7: Inicio del proceso de depuración

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.

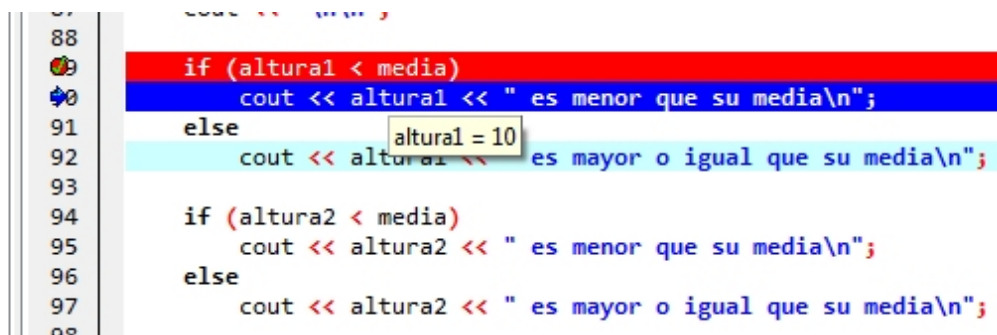


Figura 8: Inspeccionando el valor de una variable

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).

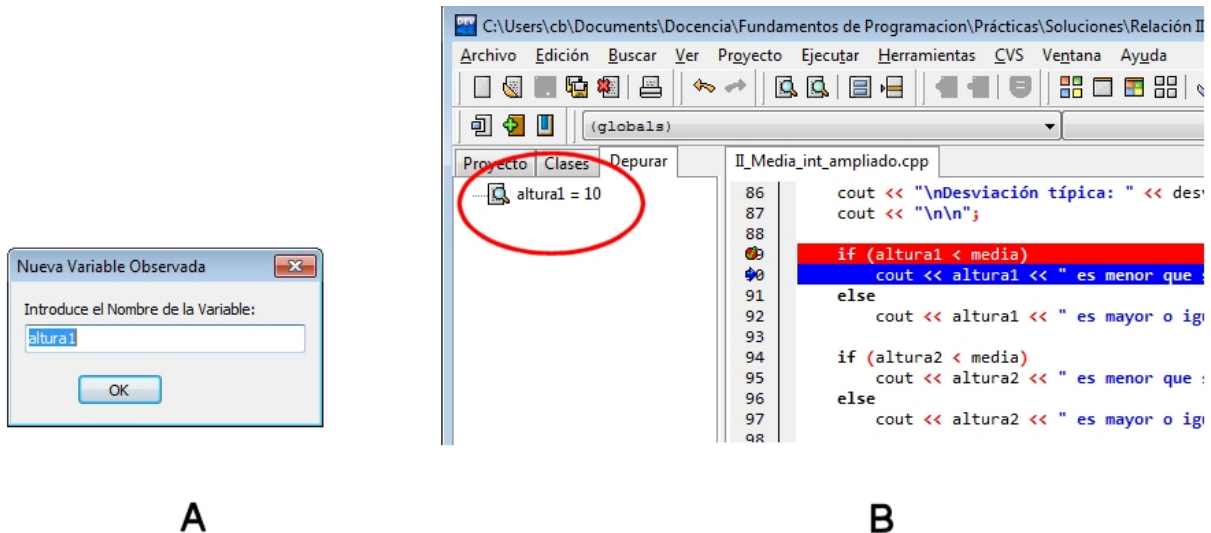


Figura 9: Añadiendo una variable para su inspección permanente

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".



Sesión 4

► **Actividades a realizar en casa**

Actividad: Resolución de problemas.

Resolved los siguientes problemas de la relación II.

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

- **Obligatorios:**
 - 2 (Subir sueldo)
 - 4 (Tres valores ordenados)
 - 8 (Año bisiesto)
 - 5 (Pasar de mayúscula a minúscula)
 - 6 (Pasar de mayúscula a minúscula y viceversa)
- **Opcionales:**
 - 10 (Tarifa aérea)

► **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). 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 derecha del ratón "Abrir Símbolo del Sistema"². Introducimos la instrucción siguiente:

```
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.

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`

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*. Por ello, la sentencia `system("pause")` no detiene el programa tal y como queríamos.

Resolved el ejercicio 9 (Renta bruta y neta) de la Relación de Problemas II. Este ejercicio entrará en la lista de problemas a resolver para la sesión 5.

Sesión 5

Estructura Repetitiva. Bucles while

► **Actividades a realizar en casa**

Actividad: Resolución de problemas.

Resolved los siguientes problemas de la relación II.

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

- **Obligatorios:**
 - 9 (Renta bruta y neta)
 - 17 (Filtro entrada mayúsculas)
 - 15 (Reinvertir capital e interés un número de años)
 - 16 (Reinvertir capital e interés hasta doblar cantidad inicial)
 - 12 (Divisores de un número)
 - 13 (El mínimo de varios reales leídos desde teclado)
- **Opcionales:**
 - 11 (Ventas de empresa)
 - 37 (Pinta dígitos generalizado)

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

Resolved el ejercicio 18 (número *desgarrable*) de la Relación de Problemas II. Este ejercicio estará incluido en la próxima sesión.

Sesión 6

Estructura Repetitiva: bucles for y bucles anidados

► **Actividades a realizar en casa**

Actividad: Resolución de problemas.

Resolved los siguientes problemas de la Relación de Problemas II.

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

- **Obligatorios:**
 - 18 (Número desgarrable)
 - 24 (Divisores e interés usando un for)
 - 20 (Factorial y Potencia)
 - 21 (Número combinatorio)
 - 23 (Progresión geométrica)
 - 25 (Suma de una serie)
- **Opcionales:**
 - 26 (RLE)

Sesión 7

Estructura Repetitiva: bucles for y bucles anidados

► **Actividades a realizar en casa**

Actividad: Resolución de problemas.

Resolved los siguientes problemas de la Relación de Problemas II.

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

- **Obligatorios:**
 - 27 (Interés, con varios tipos)
 - 28 (Parejas de caracteres)
 - 30, 31, 32 (Triángulo y cuadrado de números)
 - 46 (Número triangular)
 - 29 (Cuenta cifras)
- **Opcionales:**
 - 33 (Número feliz)

Sesión 8

Funciones

► **Actividades a realizar en casa**

Actividad: Resolución de problemas.

Resolved los siguientes ejercicios de la relación de problemas III:

- *Obligatorios:*
 - 1 (Errores en funciones)
 - 2 (Máximo de tres doubles)
 - 3 (Factorial y Potencia)
 - 4 (Progresión geométrica)
 - 5 (Mayúscula a minúscula y viceversa)
- *Opcionales:*
 - 6 (Gaussiana)

Actividades de Ampliación

Familiarizarse con las webs de referencias de funciones estándar.

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

<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.

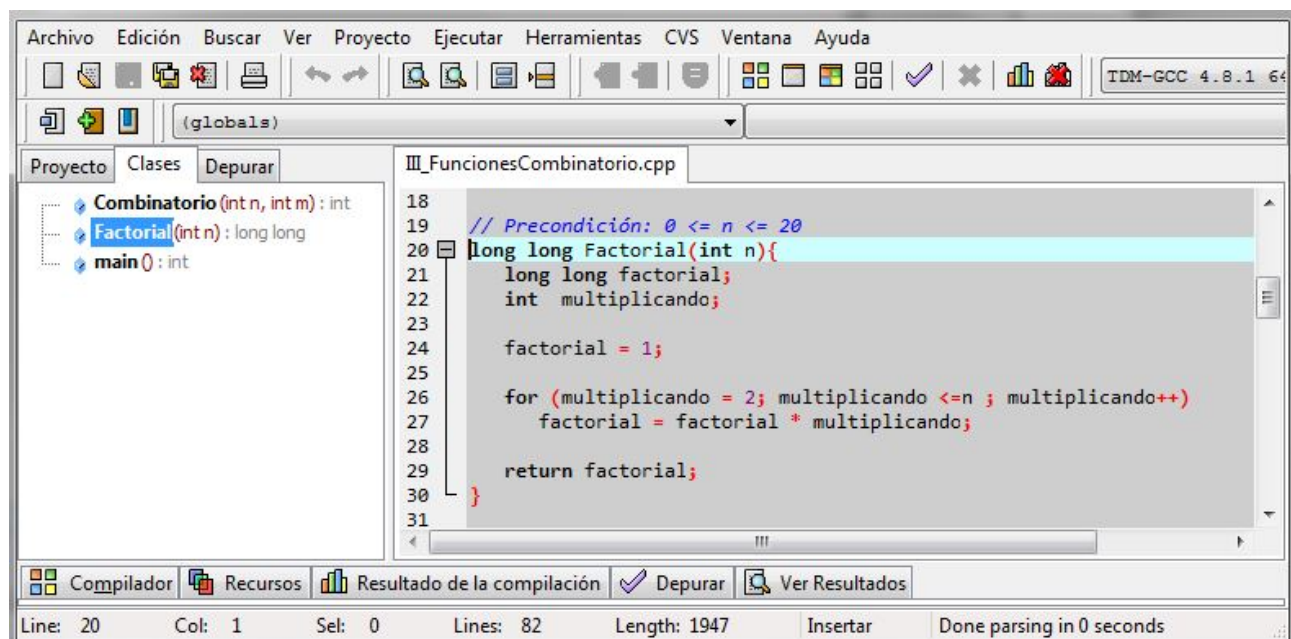


Figura 10: Explorador de clases

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?

Si ha terminado, empiece a resolver el ejercicio 7 (Recta) que tendrá que entregar en la próxima sesión de prácticas.

Sesión 9

Clases

► Actividades a realizar en casa

Actividad: Resolución de problemas.

Resolved los siguientes ejercicios de la relación de problemas III:

- *Obligatorios:*
 - 7 (Recta)
 - 8 (Progresión geométrica)
 - 10 (Interés)
 - 15 (Empresa. Sólo el diseño)
- *Opcionales:*
 - 9 (Gaussiana)
 - 16 (Empresa. Implementación)

► Actividades a realizar en las aulas de ordenadores

Inspección de objetos

En esta sesión de prácticas vamos a trabajar sobre la manera en la que Dev C++ presenta la información sobre la estructura de las clases definidas en el programa sobre el que se está trabajando y las facilidades que ofrece para monitorizar la ejecución de programas que incluyen clases.

Usaremos como ejemplo una versión simple de la clase `SegmentoDirigido` que puede encontrar en los apuntes de teoría. En primer lugar, crearemos la carpeta `III_SegmentoDirigido` en `U:\FP` y copiaremos en ella el fichero fuente `III_SegmentoDirigido.cpp` (disponible en decsai).

Emplearemos de nuevo el *explorador de clases*: seleccionar `Ver | Ir al Explorador de Clases` (ó simplemente `Ctrl+F3`). En la figura 11

puede observar que el explorador de clases muestra, para este programa, información acerca de los componentes de la clase `SegmentoDirigido`, definida en el fichero fuente abierto en el editor. También muestra las cabeceras de las funciones definidas en el fichero, como ya conocemos (en este caso, únicamente la función `main`).

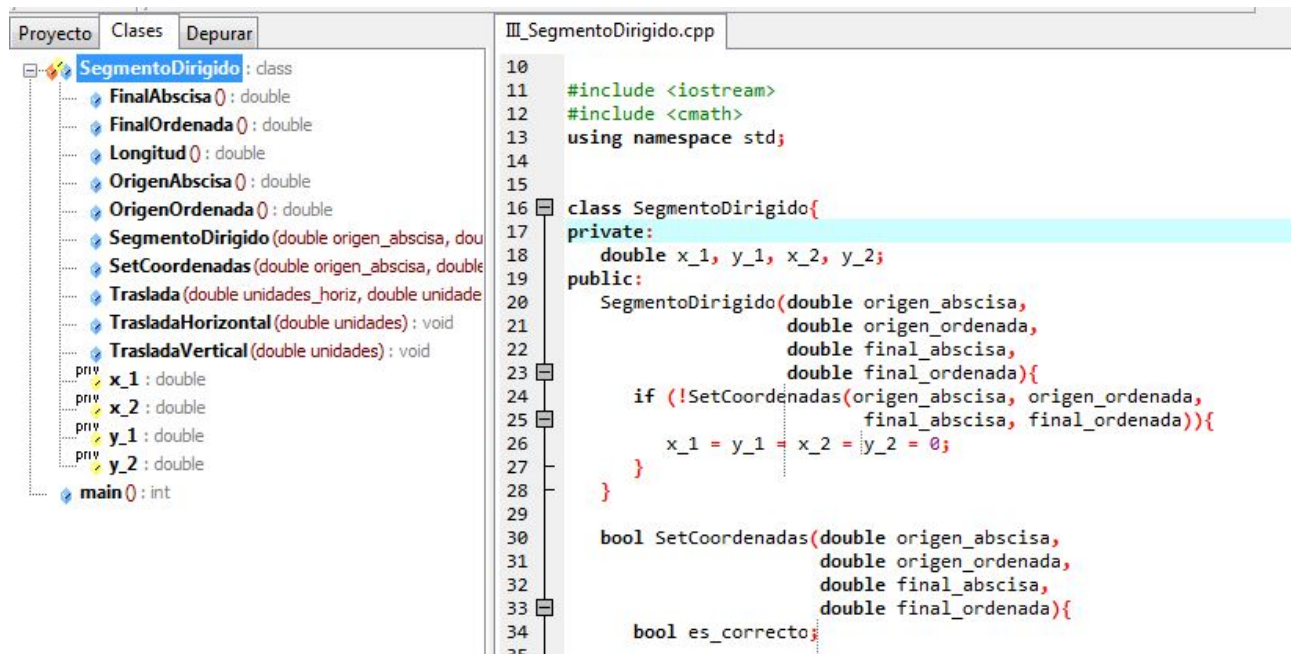


Figura 11: Explorador de clases

En primer lugar observe que el explorador de clases organiza los elementos que muestra de manera jerárquica, por niveles. En este ejemplo encontramos dos elementos en el nivel principal (por orden de aparición en el explorador de clases):

- La clase `SegmentoDirigido`
- La función `main`.

Se etiqueta al elemento `SegmentoDirigido` con la palabra `class`. Observe que se enumeran una serie de elementos dentro de esa clase que gráficamente se muestran asociados a `SegmentoDirigido` mediante líneas.

Dev C++ organiza la presentación de los elementos de la clase enumerando en primer lugar los *métodos* (por orden alfabético) y a continuación los *datos* (también por orden alfabético). Haciendo click sobre el nombre de cualquier componente de la clase el editor muestra el código de la función seleccionada (su implementación o definición), o se accede a la declaración del dato seleccionado. Es una manera rápida de acceder al código de cualquier elemento de una clase.

Dev C++ usa un conjunto de iconos para mostrar los diferentes elementos del programa:



Los objetos, como cualquier otro dato, pueden ser monitorizados durante la depuración de un programa. Cuando se añade un objeto para su inspección se muestran sus campos y sus valores. Por ejemplo, en la figura 12 se muestra el contenido del objeto `un_segmento` después de haber asignado los valores a sus cuatro campos.

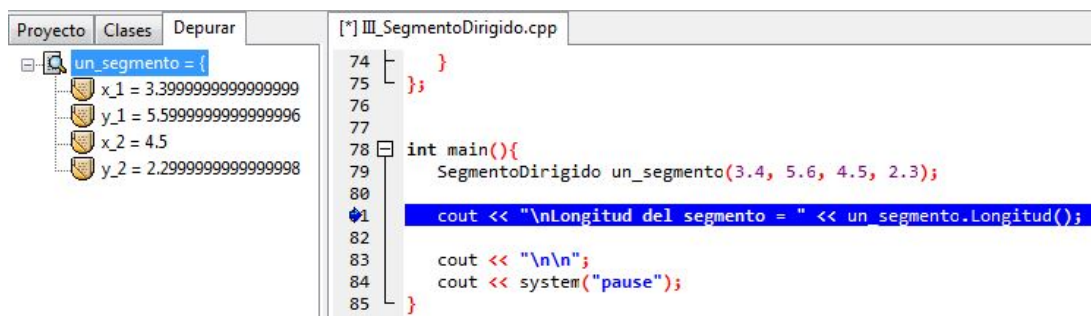


Figura 12: Explorador de clases mostrando el contenido de un objeto `SegmentoDirigido`

Para llegar al estado mostrado en la figura 12:

1. se estableció un punto de ruptura después de crear el objeto,
2. se añadió la variable `un_segmento` a la lista de datos monitorizados (botón derecho sobre `un_segmento` y `Añadir watch`)
3. se inició la depuración del programa (`Ejectuar` | `Depurar`).

Una vez que aparece el objeto `un_segmento` entre los datos monitorizados puede trabajar con él (y con cualquiera de los campos que lo forman) de la misma manera que con cualquier otro dato monitorizado.

Durante la depuración puede *avanzar paso a paso* (F8), y entrar a ejecutar los métodos de la clase línea a línea, tal y como se vio en la sesión de prácticas con funciones. Por ejemplo, establezca un punto de ruptura en la línea

```
un_segmento.TrasladaHorizontal (10);
```

continúe la depuración hasta llegar a esa línea y pulse F8. Cuando se está ejecutando el método `TrasladaHorizontal`, los cambios realizados por éste en los datos miembro no aparecen reflejados en la ventana de inspección del objeto `un_segmento` (para comprobarlo es posible que tenga que mover el ratón sobre el código para que se refresque la

ventana de inspección) Si estamos ejecutando un método y queremos ver cómo cambian los datos miembro, debemos añadir explícitamente una inspección para los datos miembros (ver figura 13).

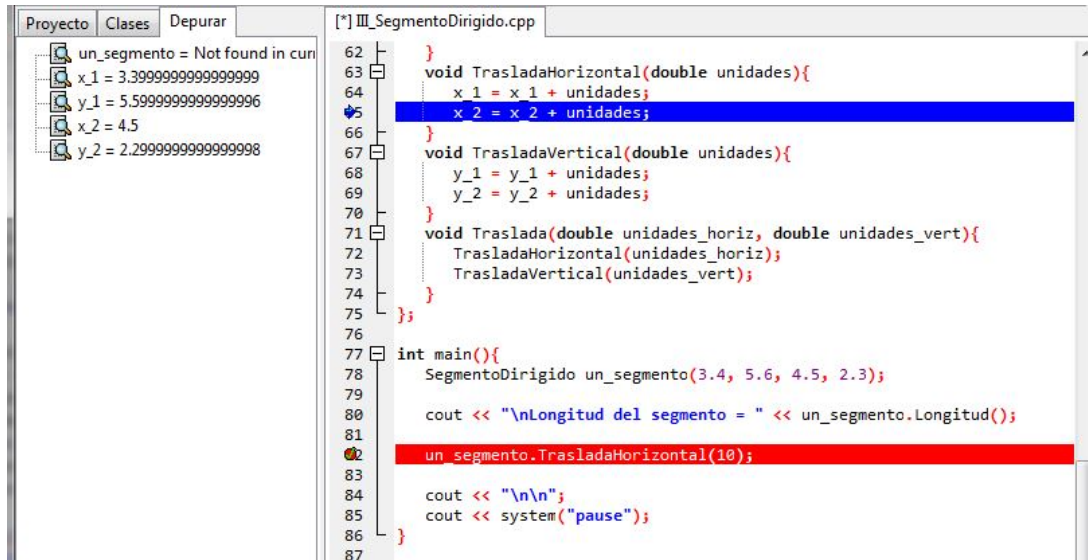


Figura 13: Explorador de clases mostrando los datos miembro durante la depuración

Al salir del método y volver a la función `main` comprobaremos que el objeto `un_segmento` se actualiza y se muestran los valores correctos.

Sesión 10

Vectores

► **Actividades a realizar en casa**

Actividad: Resolución de problemas.

Resolved los siguientes ejercicios de las relaciones de problemas III y IV:

- *Obligatorios:*
 - 13 (III. Nómina Empresa)
 - 1 (IV. Palíndromo)
 - 2 (IV. Modifica componente)
 - 3 (IV. Intercambia e Invierte)
- *Opcionales:*
 - 14 (III. Bicicleta)
 - 4 (IV. Elimina Componente)

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

Sesión 11

► **Actividades a realizar en casa**

Actividad: Resolución de problemas.

Resolved los siguientes ejercicios de la relación de problemas IV:

- *Obligatorios:*
 - 5 (Elimina mayúsculas)
 - 19 (Número de secuencias ascendientes)
 - 9 (Cuenta mayúsculas)
 - 10 (Clase Contador de mayúsculas)
 - 7 (Elimina repetidos, sólo apartados a) y b))
- *Opcionales:*
 - 6 (Elimina mayúsculas eficiente)

Sesión 12

► **Actividades a realizar en casa**

Actividad: Resolución de problemas.

Resolved los siguientes ejercicios de la relación de problemas IV:

- *Obligatorios:*
 - 7 (Elimina repetidos eficiente, apartado c))
 - 16 (Login automático)
 - 15 (Eratostenes)
 - 13 (Matrices. Sólo apartados a) Traspuesta y d) Multiplicación)
- *Opcionales:*
 - 13 (Matrices. Apartado b) Máximo de los mínimos de cada fila)



Fundamentos de Programación.

Relaciones de Problemas.

RELACIÓN DE PROBLEMAS I. Introducción a C++

Problemas Básicos

1. Crear 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.

2. Un banco presenta la siguiente oferta. Si se deposita una cantidad de euros `capital` durante un año a plazo fijo, se dará un interés dado por la variable `interes`. Realizad un programa que lea una cantidad `capital` y un interés `interes` desde teclado y calcule en una variable `total` el dinero que se tendrá al cabo de un año, aplicando la fórmula:

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

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`.

A continuación, el programa debe imprimir en pantalla el valor de la variable `total`. Tanto el `capital` como el `interes` serán valores reales. 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.

Supongamos que queremos modificar la variable original `capital` con el nuevo valor de `total`. ¿Es posible hacerlo directamente en la expresión de arriba?

Nota: El operador de división en C++ es /

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

3. Cread un programa que nos pida la longitud del radio, calcule el área del círculo y la longitud de la circunferencia correspondientes, y nos muestre los resultados en pantalla. Recordad que:

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

Usad el literal 3.1416 a lo largo del código, cuando se necesite multiplicar por π .

Una vez hecho el programa, cambiad las apariciones de 3.1416 por 3.14159, re-compilad y ejecutad (La parte de compilación y ejecución se realizará cuando se vea en clase de prácticas el entorno de programación).

RELACIÓN DE PROBLEMAS I. Introducción a C++

¿No hubiese sido mejor declarar un dato *constante* PI con un valor igual a 3.14159, y usar dicho dato donde fuese necesario? Hacedlo tal y como se explica en las transparencias de los apuntes de clase.

Cambiad ahora el valor de la constante PI por el de 3.1415927, recompilad y ejecutad.

Finalidad: Entender la importancia de las constantes. Dificultad Baja.

- Realizar un programa que lea los coeficientes reales μ y σ de una función gaussiana (ver definición abajo). A continuación el programa leerá un valor de abscisa x y se imprimirá el valor que toma la función en x

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

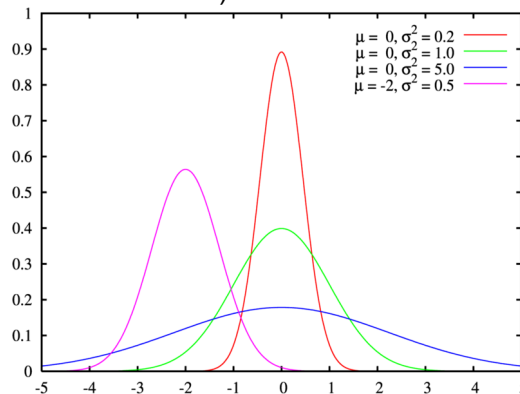
El parámetro μ se conoce como *esperanza* o *media* y σ como *desviación típica* (*mean* y *standard deviation* en inglés). Para definir la función matemática e usad la función `exp` de la biblioteca `cmath`. En la misma biblioteca está la función `sqrt` para calcular la raíz cuadrada. Para elevar un número al cuadrado se puede usar la función `pow`, que se utiliza en la siguiente forma:

`pow(base, exponente)`

En nuestro caso, el exponente es 2 y la base $\frac{x-\mu}{\sigma}$. Comprobad que los resultados son correctos, usando el applet disponible en

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

o bien algunos de los ejemplos de la figura siguiente (observad que el valor de la desviación está elevado al cuadrado):



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

- Las ganancias de un determinado producto se reparten entre el diseñador y los tres fabricantes del mismo. Diseñar 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. Utilizad el tipo `double` para todas las variables.

Importante: No repetid cálculos ya realizados.

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

6. Queremos realizar un programa para intercambiar los contenidos de dos variables enteras. El programa leerá desde teclado dos variables `edad_Pedro` y `edad_Juan` e intercambiará sus valores. A continuación, mostrará en pantalla las variables ya modificadas. El siguiente código no funciona correctamente.

```
edad_Pedro = edad_Juan;  
edad_Juan = edad_Pedro;
```

¿Por qué no funciona? Buscad una solución.

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

7. Escribid 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:

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

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

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

8. Escribir un programa que lea un valor entero. Supondremos que el usuario introduce siempre un entero de tres dígitos, como por ejemplo 351. Escribid en pantalla los dígitos separados por tres espacios en blanco. Con el valor anterior la salida sería:

```
3    5    1
```

Dificultad Baja.

9. Leed desde teclado tres variables correspondientes a un número de horas, minutos y segundos, respectivamente. Diseñar un algoritmo que calcule las horas, minutos y

segundos dentro de su rango correspondiente. Por ejemplo, dadas 10 horas, 119 minutos y 280 segundos, debería dar como resultado 12 horas, 3 minutos y 40 segundos. El programa no calculará meses, años, etc sino que se quedará en los días.

Como consejo, utilizad el operador / que cuando trabaja sobre datos enteros, representa la división entera. Para calcular el resto de la división entera, usad el operador %.

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

10. Indicar si se produce un problema de precisión o de desbordamiento en los siguientes ejemplos indicando 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. Hacedlo escribiendo la sentencia `cout.precision(numero_digitos);` en cualquier sitio del programa antes de la ejecución de `cout << real1 << ", " << real2;`. Hay que destacar que al trabajar con reales siempre debemos asumir representaciones aproximadas por lo que no podemos pensar que el anterior valor `numero_digitos` esté indicando un número de decimales con representación exacta.

- a)

```
int chico, chico1, chico2;
chico1 = 123456789;
chico2 = 123456780;
chico = chico1 * chico2;
```
- b)

```
long grande;
int chico1, chico2;
chico1 = 123456789;
chico2 = 123456780;
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.

11. Realizar un programa que declare las variables x , y y z , les asigne los valores 10, 20 y 30 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).

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

12. Realizad 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.

13. Realizad 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.

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

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

15. Supongamos el siguiente código:

```
int entero;
char character;

character = '7';
entero = character;
```

La variable `entero` almacenará el valor 55 (el orden en la tabla ASCII del carácter '7'). Queremos construir una expresión que devuelva el entero 7, para asignarlo a la variable `entero`. Formalmente:

Supongamos una variable `car` de tipo carácter que contiene un valor entre '0' y '9'. Construid un programa que obtenga el correspondiente valor entero, se lo asigne a una variable de tipo `int` llamada `entero` y lo imprima en pantalla. Por ejemplo, si la variable `car` contiene '7' queremos asignarle a `entero` el valor numérico 7.

Nota. La comilla simple para representar un literal de carácter es la que hay en el teclado del ordenador debajo de la interrogación ?.

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

16. Razonar 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.*

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

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

Escribid una expresión lógica que nos informe cuando 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.

Escribid un programa que lea las variables `letra`, `edad` y `año`, calcule el valor de las expresiones lógicas anteriores e imprima el resultado. Tened 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.

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

18. 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.

Problemas de Ampliación

19. 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. Diseñar 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.

20. Cread 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.

21. Cread 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`.

22. Declarar las variables necesarias y traducir las siguientes fórmulas a expresiones válidas del lenguaje C++.

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

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

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

Algunas funciones de
cmath

$\text{sen}(x) \rightarrow \sin(x)$

$\cos(x) \rightarrow \cos(x)$

$x^y \rightarrow \text{pow}(x, y)$

$\ln(x) \rightarrow \log(x)$

$e^x \rightarrow \exp(x)$

Dificultad Baja.

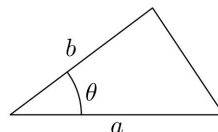
23. Dos locomotoras parten de puntos distintos avanzando en dirección contraria sobre la misma vía. Se pide redactar un programa para conocer las distancias que habrán recorrido ambas locomotoras antes de que choquen teniendo en cuenta que la primera locomotora viaja a una velocidad constante V_1 , que 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.

24. 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 $A = \frac{1}{2}ab \sin(\theta)$. Construid 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.

25. 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

Problemas Básicos

1. Ampliad el ejercicio 7 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 independientes. *Dificultad Baja.*

2. Cread un programa que lea el valor de la edad (dato de tipo entero) y salario (dato de tipo real) de una persona. Subid el salario un 5% si éste es menor de 300 euros y la persona es mayor de 65 años. Imprimid el resultado por pantalla. En caso contrario imprimid el mensaje "No es aplicable la subida". En ambos casos imprimid el salario resultante.

Realizad el mismo ejercicio pero subiendo el salario un 4% si es mayor de 65 o menor de 35 años. Además, si también tiene un salario inferior a 300 euros, se le subirá otro 3%.

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

3. Realizar 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. Escribid 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.

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

5. 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 en una variable llamada `letra_convertida`.

En cualquier otro caso, 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`.

a) Se propone una primera solución como sigue:

```
char letra_convertida, letra_original;
const int DISTANCIA_MAY_MIN = 'a'-'A';

cout << "\nIntroduzca una letra --> ";
cin >> letra_original;

if ((letra_original >= 'A') && (letra_original <= 'Z')){
    letra_convertida = letra_original + DISTANCIA_MAY_MIN;
    cout << letra_convertida;
}
else{
    cout << letra_original << " no es una mayúscula";
}
```

El problema es que dentro de la misma sentencia condicional se realizan cálculos (`letra_convertida = letra_original + DISTANCIA_MAY_MIN`) y salidas de resultados en pantalla (`cout << letra_convertida;`). Para evitarlo, se propone el uso de una variable que nos indique lo sucedido en el bloque de cálculos.

b) Usamos en primer lugar un `string`:

```
string tipo_letra;
.....
if ((letra_original >= 'A') && (letra_original <= 'Z'))
    tipo_letra = "es mayúscula";

if (tipo_letra == "es mayúscula")
    letra_convertida = letra_original + DISTANCIA_MAY_MIN;
else
    letra_convertida = letra_original;

cout << "\nEl carácter " << letra_original
    << " una vez convertido es: " << letra_convertida;
```

Nota. Para poder usar el operador de comparación `==` entre dos `string`, hay que incluir la biblioteca `string`.

Si se opta por esta alternativa, el suspenso está garantizado. ¿Por qué?

- c) Mucho mejor si usamos una variable lógica llamada `es_mayuscula`, de la siguiente forma:

```
if ((letra_original >= 'A') && (letra_original <= 'Z'))
    es_mayuscula = true;

if (es_mayuscula)
    letra_convertida = letra_original + DISTANCIA_MAY_MIN;

cout << "\nEl carácter " << letra_original
      << " una vez convertido es: " << letra_convertida;
```

Sin embargo, hay un error lógico ya que la variable `es_mayuscula` se podría quedar sin un valor asignado (en el caso de que la expresión lógica fuese `false`). El compilador lo detecta y da el aviso al inicio de la sentencia `if (es_mayuscula)`. Comprobadlo para familiarizarnos con este tipo de avisos y proponer una solución.

Moraleja: Hay que garantizar la asignación de las variables lógicas, en todos los casos posibles

Finalidad: Mostrar cómo se comunican los distintos bloques de un programa a través de variables que indican lo que ha sucedido en el bloque anterior y destacar la importancia de incluir un bloque `else`, para garantizar que siempre se ejecuta el código adecuado, en todas las situaciones posibles. Dificultad Baja.

6. Queremos modificar el ejercicio 5 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`

Para ello, añadimos una variable nueva `es_minuscula` para detectar el caso en el que la letra sea una minúscula. Si escribimos el siguiente código

```
if (letra_original >= 'A') && (letra_original <= 'Z')
    es_mayuscula = true;
else
    es_minuscula = true;
```

estaremos cometiendo el tipo de error indicado en el ejercicio 5, ya que si le asignamos un valor a una de las variables lógicas, no se lo asignamos a la otra y se queda con un valor indeterminado. Para resolverlo, planteamos la siguiente solución:

```
if ((letra_original >= 'A') && (letra_original <= 'Z')){
    es_mayuscula = true;
    es_minuscula = false;
}
else{
    es_mayuscula = false;
    es_minuscula = true;
}
```

o si se prefiere, de una forma más concisa:

```
es_mayuscula = (letra_original >= 'A') &&
               (letra_original <= 'Z');
es_minuscula = !es_mayuscula;
```

En este planteamiento hay un error lógico que se comete de forma bastante habitual, cuando se quiere detectar más de dos situaciones posibles. En la asignación

```
es_minuscula = !es_mayuscula;
```

estamos diciendo que una minúscula es todo aquello que no sea una mayúscula. Esto no es correcto, ya que un símbolo como # no es ni mayúscula ni minúscula. Deberíamos haber usado lo siguiente:

```
es_mayuscula = (letra_original >= 'A') &&
               (letra_original <= 'Z');
es_minuscula = (letra_original >= 'a') &&
               (letra_original <= 'z');
```

Completad el ejercicio, asignándole el valor correcto a la variable `letra_convertida` e imprimid en pantalla el valor de `letra_convertida`.

Finalidad: Plantear una estructura condicional anidada. Dificultad Baja.

7. Modificad la solución del ejercicio 2 (subida salarial) para que no se mezclen E/S (entradas y salidas) con cálculos dentro de la misma estructura condicional.

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

8. Cread 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.

Dificultad Baja.

9. Cread 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.

RELACIÓN DE PROBLEMAS II. Estructuras de Control

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

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

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 modificación se hará de la siguiente forma:

- Se baja un 3 % la retención fiscal a los autónomos
- Para los no autónomos:
 - Se sube un 1 % la retención fiscal a todos los pensionistas.
 - Al resto de trabajadores se le sube un 2 % la retención fiscal. Una vez hecha esta subida lineal del 2%, se le aplica (sobre el resultado anterior) las siguientes subidas adicionales, dependiendo de su estado civil y niveles de ingresos:
 - Se sube un 6 % la retención fiscal si la renta bruta es menor de 20.000 euros
 - Se sube un 8 % la retención fiscal a los casados con renta bruta superior a 20.000 euros
 - Se sube un 10 % la retención fiscal a los solteros con renta bruta superior a 20.000 euros

Nota: Cuando se pide subir un x % la retención fiscal, significa que la nueva retención fiscal será la antigua más el resultado de realizar el x % sobre la antigua retención.

$$\text{Nueva_retención} = \text{Antigua_retención} + \text{Antigua_retención} * x / 100$$

De forma análoga, si se le baja la retención, habrá que restar en vez de sumar.

Finalidad: Plantear una estructura condicional anidada. Dificultad Media.

10. 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).

RELACIÓN DE PROBLEMAS II. Estructuras de Control

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.

Cread 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)
- 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.

11. 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 ó 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.

12. Realizar 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.

13. Realizar 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. Realizad 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.

14. Realizar 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.

Realizad 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: Ejercitar el uso de bucles. Dificultad Baja.

15. Modifiquemos el ejercicio 2 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. Construid 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.

16. Sobre el mismo ejercicio del capital y los intereses, construid 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.

RELACIÓN DE PROBLEMAS II. Estructuras de Control

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

17. 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. Calculad la minúscula correspondiente e imprimidla en pantalla. No pueden usarse las funciones `tolower` ni `toupper` de la biblioteca `cctype`. Si se quiere, se puede usar como base el proyecto que resolvió el ejercicio 14 de la relación de problemas I.

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

18. 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 $101 = ()^2$. Cread un programa que lea un entero n e indique si es o no desgarrable.

Finalidad: Ejercitar los bucles. Dificultad Baja.

19. 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). Construir un programa que, dado un número entero positivo, nos indique si el número es o no narcisista.

Finalidad: Ejercitar los bucles. Dificultad Media.

20. Calcular 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`.

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

$$0! = 1$$

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

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

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

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.

22. 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:

- Modificad la solución del ejercicio 12 de forma que el filtro de entrada usado para leer la variable `tope`, se haga con un bucle pre-test `while`.
- Modificad la solución del ejercicio 15 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.

23. Supongamos una serie numérica cuyo término general es:

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

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:

- Realizad 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`.
- 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.

Cread 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 cálculos realizados en la iteración anterior. Dificultad Baja.

RELACIÓN DE PROBLEMAS II. Estructuras de Control

24. Reescribid la solución a los ejercicios 12 (divisores) y 15 (interés) usando un bucle `for`

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

25. 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 cálculos realizados en la iteración anterior. Dificultad Media.

26. 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). Realizar 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.

27. Sobre la solución del ejercicio 15 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.

28. Escribid un programa que lea cuatro valores de tipo char (min_izda, max_dcha, min_dcha, max_dcha) e imprima las parejas que pueden formarse con los conjuntos {min_izda ... max_izda} y {min_dcha ... max_dcha}. Por ejemplo, si min_izda = b, max_izda = d, min_dcha = j, max_dcha = m, el programa debe imprimir lo siguiente:

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

Finalidad: Ejercitar los bucles anidados. Dificultad Baja.

29. (*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: Ejercitar los bucles anidados. Dificultad Baja.

30. Cread 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: Ejercitar los bucles anidados. Dificultad Baja.

31. Cread 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: Ejercitar los bucles anidados. Dificultad Media.

32. Modificad 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: Ejercitar los bucles anidados. Dificultad Media.

33. 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: Ejercitar los bucles anidados. Dificultad Media.

Problemas de Ampliación

34. Vamos a modificar el ejercicio 3 de la siguiente forma. Queremos leer dos valores enteros desde teclado y, en el caso de que uno cualquiera de ellos divida al otro, el programa nos debe decir quién divide a quién.

- a) En primer lugar, resuelve el ejercicio mezclando entradas, cálculos y salidas de resultados
- b) En segundo lugar, se pide resolver el ejercicio sin mezclar C/E,S. Para ello, se ofrecen varias alternativas. ¿Cuál sería la mejor? Escoge una e implementa la solución.

- i) Utilizar un variable de tipo `string` de la forma siguiente:

```
string quien_divide;
.....
if (a%b==0)
    quien_divide = "b divide a a" ;
.....
if (quien_divide == "b divide a a")
    cout << b << " divide a " << a;
```



Nota. Para poder usar el operador de comparación `==` entre dos `string`, hay que incluir la biblioteca `string`.

Si se opta por esta alternativa, el suspenso está garantizado. ¿Por qué?

- ii) Utilizar dos variables lógicas de la forma siguiente:

```
bool a_divide_b, b_divide_a;
.....
if (a%b==0)
    a_divide_b = true;
.....
if (a_divide_b)
    cout << a << "divide a " << b;
```

- iii) Detectamos si se dividen o no y usamos otras dos variables que me indiquen quien es el dividendo y quien el divisor:

```
bool se_dividen;
int Divdo, Dvsor;
.....
if (a%b==0){
    Divdo = a;
.....
if (se_dividen)
    cout << Dvsor << " divide a " << Dvdo;
```

RELACIÓN DE PROBLEMAS II. Estructuras de Control

Completar la solución elegida para contemplar también el caso en el que alguno de los valores introducidos sea cero, en cuyo caso, ninguno divide al otro.

Dificultad Media.

35. Modificad el ejercicio 4 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, se recomienda usar una variable de tipo enumerado.

Finalidad: Usar el tipo enumerado para detectar cuándo se produce una situación determinada. Dificultad Baja.

36. En el ejercicio 6 se han usado dos variables lógicas (`es_mayuscula`, `es_minuscula`), que nos proporciona la distinción entre 4 casos distintos (VV, VF, FV, FF). Sin embargo, sólo queremos distinguir tres posibilidades, a saber, es mayúscula, es minúscula y no es un carácter alfabético. Para ello, volved a resolver el ejercicio 6 sustituyendo las dos variables lógicas por un tipo enumerado adecuado.

Finalidad: Usar el tipo enumerado para detectar cuándo se produce una situación determinada. Dificultad Media.

37. En el ejercicio 8 de la Relación de Problemas I se pedía escribir un programa que le-yese un valor entero de tres dígitos e imprimiese los dígitos separados por un espacio en blanco. Haced 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 cálculos.

Dificultad Media.

38. 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.

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

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

RELACIÓN DE PROBLEMAS II. Estructuras de Control

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.

40. 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.

41. 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.

42. 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.

43. 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.

44. 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.

45. Diseñar un programa para jugar a adivinar un número. 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 considerad a) que haya acertado o b) se haya hartado y decida terminar (escoged cómo se quiere que se especifique esta opción)

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

Dificultad Media.

46. 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 t ope introducido desde teclado. Para ello, debe ir probando la suma de todas las posibles secuencias de enteros menores que el número dado. Hay que calcular explícitamente la suma con un bucle, por lo que no puede aplicarse la fórmula que nos da la sumatoria de los n primeros valores, a saber, $n(n + 1)/2$

Dificultad Baja.

47. Escribir un programa que lea una secuencia de números enteros en el rango de 0 a 100 terminada en un número mayor que 100 o menor que 0 y encuentre 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:

23 25 7 40 45 45 73 73 71 4 9 101

el programa nos debe indicar que la mayor subsecuencia empieza en la posición 3 (en el 7) y tiene longitud 6 (termina en la segunda aparición del 73)

Dificultad Media.

48. 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 multiplicando por 2 el multiplicador m y dividiendo (sin decimales) por dos el 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

RELACIÓN DE PROBLEMAS II. Estructuras de Control

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 $37 \cdot 12 = 12 + 48 + 384 = 444$

Cread un programa para leer dos enteros n y m y calcule su producto utilizando este algoritmo.

Dificultad Media.

49. Construid 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:	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 @	
	#	

En este caso, sí se encuentra.

Dificultad Media.

50. 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.

51. 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.

52. 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 $a_n = \frac{fib(n+1)}{fib(n)}$ siendo $fib(n)$ el término n -ésimo de la sucesión de fibonacci que se define como:

$$fib(0) = 0, fib(1) = 1 \text{ y } fib(n) = fib(n-2) + fib(n-1) \text{ para } n \geq 2$$

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 $n = 3$

Dificultad Media.

RELACIÓN DE PROBLEMAS III. Funciones y Clases

Problemas Básicos

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

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

Construid un programa principal que llame a dicha función con unos valores leídos desde teclado. Supongamos que dichos valores los leemos con `cin` dentro de la propia función, en vez de hacerlo en el `main`. El suspenso está garantizado ¿Por qué?

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

3. Reescribid la solución del ejercicio 20 (factorial y potencia) de la Relación de Problemas II, modularizándola con funciones.

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

4. Cread las siguientes funciones relacionadas con la progresión geométrica que se vio en el ejercicio 23 de la relación de problemas II. Analizad 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.

Cread un programa principal que llame a dicha función.

- b) Cambiemos ahora la implementación de la anterior 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 3 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`.

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

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

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

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

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

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

5. Recuperad la solución del ejercicio 36 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';
```

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 mayúscula, devuelve el carácter pasado como argumento.

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

6. En el ejercicio 4 de la relación de problemas I (página [RP-I.2](#)) se vio cómo obtener el valor de ordenada asignado por la función gaussiana, sabiendo el valor de abscisa x . Recordemos que esta función matemática dependía de dos parámetros μ (esperanza) y σ (desviación) y venía definida por:

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

Cread un programa que lea un valor de esperanza y desviación y a continuación lea un número entero n que indique el número de abscisas que se van a procesar. Leed un total de n valores e imprimid en pantalla el valor de la función gaussiana en dichos valores. El cómputo de la gaussiana debe hacerse en una función.

Ahora estamos interesados en obtener el área que cubre la función en el intervalo $[-\infty, x]$. Dicho valor se conoce como la *distribución acumulada (cumulative distribution function)* en el punto x , abreviado $CDF(x)$. Matemáticamente se calcula realizando la integral:

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

tal y como puede comprobarse ejecutando el applet disponible en:

<http://danielsoper.com/statcalc3/calc.aspx?id=53>

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

$$CDF(x) = \text{Área hasta } (x) \approx 1 - \text{gaussiana}(x)(b_1t + b_2t^2 + b_3t^3 + b_4t^4 + b_5t^5)$$

dónde:

$$t = \frac{1}{1 + b_0x} \quad b_0 = 0,2316419 \quad b_1 = 0,319381530 \quad b_2 = -0,356563782$$

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

Cread otra función para calcular el área hasta un punto cualquiera x , es decir, $CDF(x)$, usando la anterior aproximación. Modificad el programa principal para que también imprima los valores de CDF correspondientes a los enteros leídos.

Finalidad: Familiarizarnos con la definición de funciones y el paso de parámetros. Entender la importancia de la ocultación de información. Dificultad Baja.

Problemas sobre clases

7. 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?

Añadir 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 y lea un valor de ordenada e imprima la abscisa que le corresponde. Hacedlo 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.



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.

Observad 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*

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 en el constructor y en el método `SetCoeficientes` y realice las operaciones que estime oportuno en el caso de que los coeficientes pasados como parámetros no sean correctos.

8. En el ejercicio 4 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.

RELACIÓN DE PROBLEMAS III. Funciones y Clases

- 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.

9. Recuperad el ejercicio 6 de esta relación de problemas sobre la función gaussiana. En vez de trabajar con funciones, plantead la solución con una clase.

Dificultad Media.

10. Se quiere construir una clase `DepositoSimulacion` para simular préstamos, ofreciendo la funcionalidad descrita en los ejercicios 15 (reinvierde capital e interés un número de años) y 16 (reinvierde capital e interés hasta obtener el doble de la cantidad inicial) de la relación de problemas II (página RP-II.7). 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.

11. Recuperad la solución del ejercicio 9 (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, definí una clase `MenuSiNO` que encapsule esta funcionalidad y cambiar el programa principal para que use esta clase.

12. Recuperad la solución del ejercicio 7 (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.

13. Se quiere construir una clase `Nomina` para realizar la funcionalidad descrita en el ejercicio 12 de la relación de problemas I sobre la nómina del fabricante y diseñador (página RP-I.5). 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):

$$\text{salario_neto} = \text{salario_bruto} - \text{salario_bruto} * \text{retencion_fiscal} / 100.0$$

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
```

RELACIÓN DE PROBLEMAS III. Funciones y Clases

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.

14. 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.

15. Recuperad la solución del ejercicio 11 de la Relación de Problemas II (Empresa). Reescribid el programa principal usando una clase *Ventas* para gestionar los cálculos 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.

El programa principal sería de la siguiente forma:

```
cin >> identif_sucursal;
fichero_vacio = identif_sucursal == TERMINADOR;

if (!fichero_vacio){
    while (identif_sucursal != TERMINADOR){
        cin >> cod_producto;
        cin >> unidades_vendidas;

        --> Actualiza el número de unidades
            vendidas de la sucursal leida

        cin >> identif_sucursal;
    }

    --> Obtener el identificador y el número de ventas
        de la sucursal ganadora
}
```

Finalidad: Diseño de una clase. Dificultad Media.

16. Implementar los métodos de la clase Ventas del ejercicio anterior.

Finalidad: Diseño de una clase. Dificultad Media.

Problemas de Ampliación

17. Definid funciones aisladas para implementar las soluciones a los siguientes ejercicios de la relación de problemas II: **25** (serie), **19** (narcisista), **33** (feliz).

Dificultad Baja.

18. 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.

19. 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.

20. Implementad la clase del ejercicio **19** de esta relación de problemas. *Dificultad Media.*

21. 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`. Includ un sencillo programa principal que llame a los métodos.

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

22. Recuperad la solución del ejercicio 16 (Empresa) y modificadlo convenientemente para que los datos miembros que referencia los identificadores de las sucursales sean constantes.

Finalidad: Trabajar con datos miembros constantes. Dificultad Baja.

23. 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.

24. Construid una clase llamada `MedidaAngulo` que represente una medida de un ángulo. Al igual que se hizo en el ejercicio 23, 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.

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

$$(x_0 - x_1)^2 + (y_0 - y_1)^2 \leq r^2$$

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

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

RELACIÓN DE PROBLEMAS IV. Vectores

Los ejercicios básicos tienen como objetivo ampliar la clase `SecuenciaCaracteres`

```
class SecuenciaCaracteres {
private:
    static const int TAMANIO = 50;
    char vector_privado[TAMANIO];
    int total_utilizados;
public:
    SecuenciaCaracteres()
        :total_utilizados(0){
    }
    int TotalUtilizados(){
        return total_utilizados;
    }
    int Capacidad(){
        return TAMANIO;
    }
    void Aniade(char nuevo){
        if (total_utilizados < TAMANIO){
            vector_privado[total_utilizados] = nuevo;
            total_utilizados++;
        }
    }
    char Elemento(int indice){
        return vector_privado[indice];
    }
};
```

Importante:

- Para todos los ejercicios, se ha de diseñar una batería de pruebas.
- 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) desde la entrada de datos por defecto.

Problemas Básicos

1. Añadid un método `EsPalindromo` a la clase `SecuenciaCaracteres` que nos diga si la secuencia 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. Incluya un programa principal que lea una serie de caracteres hasta llegar al terminador '#' y diga si es un palíndromo. Tenga en cuenta la observación de la página anterior sobre la lectura de los caracteres con `cin.get()`.

Finalidad: Método que accede a las componentes del vector. Dificultad Baja.

2. Añadid un método con la cabecera:

```
void Modifica (int indice_componente, char nuevo)
```

para que sustituya la componente con índice `indice_componente` por el valor `nuevo`.

Este método está pensado para modificar una componente ya existente, pero no para añadir componentes nuevas. Por tanto, en el caso de que la componente no esté dentro del rango correcto, el método no modificará nada.

Finalidad: Ejemplo de método que modifica el estado del objeto. Gestión de precondiciones. Dificultad Baja.

3. Añadid 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?

Añadid otro método llamado `Invierte` para invertir la secuencia. Si contenía, por ejemplo, los caracteres {'m', 'u', 'n', 'd', 'o'}, después de llamar al método se quedará con {'o', 'd', 'n', 'u', 'm'}. Implementad el método `Invierte` llamando a `IntercambiaComponentes`.

Imprimid las componentes de la secuencia desde el `main`, antes y después de llamar a dicho método. Observad que se repite el mismo tipo de código cuando se imprimen las componentes de la secuencia. Ya lo arreglaremos en el tema siguiente.

Finalidad: Ejemplo de método que modifica el estado del objeto. Gestión de precondiciones. Dificultad Baja.

4. Añadid el método `Elimina` para eliminar el carácter que se encuentre en una determinada posición. Si la secuencia contenía, por ejemplo, los caracteres

{'h','o','l','a'}, después de eliminar la componente con índice 2 (la tercera) se quedará con {'h','o','a'}. Para borrar una componente, se *desplazan* una posición a la izquierda todas las componentes que hay a su derecha.

¿Qué debería hacer el método si el índice no es correcto?

Finalidad: Ejemplo de método que modifica el estado del objeto. Gestión de precondiciones. Dificultad Baja.

5. Añadid el método `EliminaMayusculas` para eliminar todas las mayúsculas. Por ejemplo, después de aplicar dicho método al vector {'S','o','Y',' ','y','O'}, éste debe quedarse con {'o',' ','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 una mayúscula, borrarla
```

Queremos implementarlo llamando al método `Elimina` que se ha definido en el ejercicio 4 de esta relación de problemas:

```
class SecuenciaCaracteres{
    .....
    void EliminaMayusculas(){
        for (int i=0; i<total_utilizados; i++)
            if (isupper(vector_privado[i]))
                Elimina(i);
    }
};
```

El anterior código tiene un fallo. ¿Cuál? Probarlo con cualquier secuencia que tenga dos mayúsculas consecutivas, proponer una solución e implementarla.

Finalidad: Recorrido sencillo de un vector con dos bucles anidados. Dificultad Baja.

6. El algoritmo del ejercicio 5 es muy ineficiente ya que requiere trasladar muchas veces muchas posiciones (usa dos bucles anidados).

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 una mayúscula, simplemente avanzaremos `posicion_lectura`. Por el contrario, si fuese una minúscula, la colocaremos en la posición 3 y avanzaremos una posición ambas variables.

Implementad este algoritmo.

*Finalidad: Modificar un vector a través de dos **apuntadores**. Dificultad Media.*

7. Añadid 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', 'a', 'c', 'a', 'a', 'a', 'g' }
después de quitar los repetidos, se quedaría como sigue:
{ 'b', 'a', 'h', 'c', 'g' }

Implementad los siguientes algoritmos para resolver este problema:

- a) Usad un **vector local** `sin_repetidos` en el que almacenamos una única aparición de cada componente:

```
Recorrer todas las componentes de la secuencia original
Si la componente NO está en "sin_repetidos",
    añadirla (al vector "sin_repetidos")
Asignar las componentes de "sin_repetidos" a la secuencia
```

- 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. Si una componente está repetida, **se borrará de la secuencia**. Para borrar una componente, podríamos desplazar una posición a la izquierda, todas las componentes que estén a su derecha. Implementad el siguiente algoritmo:

```
Recorrer todas las componentes de la secuencia original
Si la componente se encuentra en alguna posición
    anterior, la eliminamos desplazando hacia la
    izquierda todas las componentes que hay a su derecha.
```

- c) El anterior algoritmo nos obliga a desplazar muchas componentes cada vez que encontremos una repetida. Proponed una alternativa (sin usar vectores locales) para que el número de desplazamientos sea el menor posible e implementadla.
Consejo: Usad la misma técnica que se indicó en el ejercicio 6 de eliminar las mayúsculas.

Finalidad: Usar un vector local. Modificar un vector a través de dos apuntadores. Dificultad Media.

8. Añadid 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.

Finalidad: Recorrido de las componentes de un vector, en el que hay que recordar lo que ha pasado en la iteración anterior. Dificultad Media.

9. En este ejercicio no hay que definir ninguna clase. Todas las operaciones se realizan directamente en el `main`.

Realizad un programa que vaya leyendo caracteres hasta que se encuentre un punto '.'. Contad el número de veces que aparece cada una de las letras mayúsculas e imprimirlo en pantalla.

Una posibilidad sería declarar un vector `contador_mayusculas` y conforme se va leyendo cada carácter, ejecutar un conjunto de sentencias del tipo:

```
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.

10. Sobre el ejercicio 9, cread 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)
```

Modificad 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.

Dificultad Media.

RELACIÓN DE PROBLEMAS IV. Vectores

11. Construid una clase `CaminoComeCocos` para representar el camino seguido por el usuario en el juego del ComeCocos (Pac-Man). Internamente debe usar un vector de `char` como dato miembro privado. Tendrá métodos para subir, bajar, ir a la izquierda e ir a la derecha. Dichos métodos únicamente añadirán el carácter correspondiente 's', 'b', 'i', 'd' al vector privado.

Añadid a la clase un método `PosicionMovimientosConsecutivos` que calcule la posición donde se encuentre la primera secuencia de al menos n movimientos consecutivos iguales a uno dado (que pasaremos como parámetro al método).

Por ejemplo, en el camino de abajo, si $n = 3$ y el movimiento buscado es 's', entonces dicha posición es la 6.

{'b', 'b', 'i', 's', 's', 'b', 's', 's', 's', 's', 'i', 'i', 'd'}

Cread un programa principal que lea desde un fichero los caracteres que representan las posiciones hasta llegar a un punto ('. '), lea un carácter c y un entero n e imprima en pantalla la posición de inicio de los n movimientos iguales a c .

Dificultad Baja.

12. Cread una clase `Permutacion` para representar una permutación de enteros. Para almacenar los valores enteros usaremos como dato miembro privado un vector clásico de enteros. La clase debe proporcionar, al menos, los siguientes métodos:

- `Aniade` para añadir un número a la permutación.
- `EsCorrecta` para indicar si los valores forman una permutación correcta, es decir, que contiene todos los enteros sin repetir entre el mínimo y el máximo de dichos valores. Por ejemplo, $(2, 3, 6, 5, 4)$ es una permutación correcta pero no lo es $(7, 7, 6, 5)$ (tiene el 7 como valor repetido) ni tampoco $(7, 6, 4)$ (le falta el 5).
- `NumLecturas` para saber el número de lecturas de la permutación. Una permutación de un conjunto de enteros tiene k lecturas, si para leer sus elementos en orden creciente (de izquierda a derecha) tenemos que recorrer la permutación k veces. Por ejemplo, la siguiente permutación del conjunto $\{0, \dots, 8\}$:

4 0 8 1 2 5 3 6 7

necesita 3 lecturas. En la primera obtendríamos 0, 1, 2 y 3. En la segunda 4, 5, 6 y 7 y finalmente, en la tercera, 8.

Cread un programa principal que lea desde un fichero los valores de la permutación e imprima el número de lecturas de dicha permutación.

Dificultad Media.

13. En este ejercicio, no hay que construir ninguna clase ni función. Es un ejercicio sobre recorridos de una matriz.

RELACIÓN DE PROBLEMAS IV. Vectores

Leed desde teclado dos variables `util_filas` y `util_columnas` y leed los datos de una matriz de enteros de tamaño `util_filas` x `util_columnas`. Sobre dicha matriz, se pide lo siguiente:

- a) Calcular la traspuesta de la matriz, almacenando el resultado en otra matriz.
- b) (*Examen Septiembre 2011*) La posición de aquel elemento que sea el mayor de entre los mínimos de cada fila. Por ejemplo, dada la matriz M (3×4),

9	7	4	5
2	18	2	12
7	9	1	5

el máximo entre 4, 2 y 1 (los mínimos de cada fila) es 4 y se encuentra en la posición (0, 2).

- c) Ver si existe un valor *MaxiMin*, es decir, que sea a la vez, máximo de su fila y mínimo de su columna.
- d) Leer los datos de otra matriz y multiplicar ambas matrices (las dimensiones de la segunda matriz han de ser compatibles con las de la primera para poder hacer la multiplicación)

Problemas de Ampliación

14. 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, ...

Definid 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 Elemento(int indice)` para que devuelva el elemento `indice`-ésimo de la sucesión.

Escribid 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.Elemento(i) << " ";
```

Dificultad Media.

15. ([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 tendrá tres métodos:

- `void CalculaHasta(int n)` para que calcule los primos menores que n .
Este es el método que implementa el algoritmo de Eratóstenes. Cuando se ejecute el método, se almacenarán en un vector interno del objeto todos los primos menores que n . Debe implementarse tal y como se ha indicado anteriormente, por lo que tendrá que decidir, por ejemplo, cómo gestiona el *tachado* de los números.
- `int TotalCalculados()` que devuelva cuántos primos hay actualmente almacenados.
- `int Elemento(int indice)` para que devuelva el índice-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.Elemento(i) << " ";
```

Dificultad Media.

16. (*Examen Febrero 2013*) Se está diseñando un sistema web que recolecta 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` que tendrá como único dato miembro el tamaño `N`. Hay que definir 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`. Para probar el programa o bien lea una cadena de caracteres con `getline(cin, cadena);` o bien use directamente literales `string` en el `main`.

Dificultad Media.

17. (*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.

18. Sobre el ejercicio 17, añadid 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.

RELACIÓN DE PROBLEMAS IV. Vectores

- `void MoverPalabraFinal(int k_esima)` para desplazar la palabra k -ésima al final de la frase.

Dificultad Media.

19. (*Examen Septiembre Doble Grado 2013*) Defina la clase `SecuenciaEnteros` análoga a `SecuenciaCaracteres`. Defina lo que sea necesario para calcular el número de secuencias ascendientes del vector. Por ejemplo, el vector $\{2, 4, 1, 1, 7, 2, 1\}$ tiene 4 secuencias que son $\{2, 4\}$, $\{1, 1, 7\}$, $\{2\}$, $\{1\}$.

Dificultad Media.

20. Implementad 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:

$$\frac{\text{pos} - \text{izda}}{\text{dcha} - \text{izda}} = \frac{\text{buscado} - v[\text{izda}]}{v[\text{dcha}] - v[\text{izda}]}$$

21. En este ejercicio, no hay que construir ninguna clase ni función. Es un ejercicio sobre recorridos de una matriz.

Para ahorrar espacio en el almacenamiento de matrices cuadradas simétricas de tamaño $k \times k$ se puede usar un vector con los valores de la diagonal principal y los que están por debajo de ella. Por ejemplo, para una matriz $M = \{m_{ij}\}$ el vector correspondiente sería:

$$\{m_{11}, m_{21}, m_{22}, m_{31}, m_{32}, m_{33}, m_{41}, \dots, m_{kk}\}$$

Declarar una matriz clásica `double matriz[50][50]` en el `main`, asignarle valores de forma que sea cuadrada simétrica y construir el vector pedido. Haced lo mismo pero a la inversa, es decir, construir la matriz a partir del vector.

Dificultad Media.