



Fundamentos de Programación.

Guión de Prácticas.

Curso 2015/2016

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. En la semana número i se publicará la **Sesión i** . En dicha sesión se especifica la lista de problemas que el alumno tiene que resolver.

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

La defensa de la sesión i se hará la semana siguiente (semana $i + 1$), durante las horas de prácticas. El profesor llamará aleatoriamente a los alumnos para que defiendan dichos ejercicios (a veces explicándolos a sus compañeros) Simultáneamente a la defensa, todos los alumnos tendrán que ir realizando una serie de actividades que vienen descritas en este 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 que hay que entregar son de dos tipos:

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

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

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

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

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

Muy importante:

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

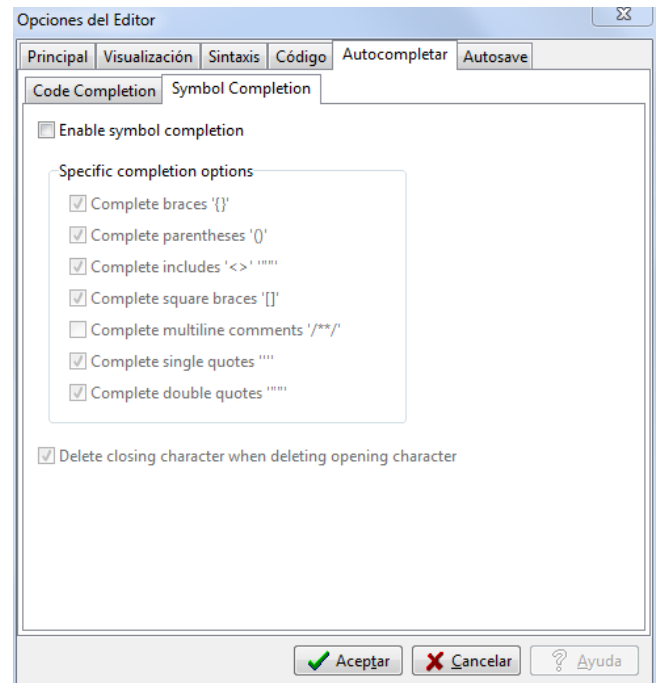
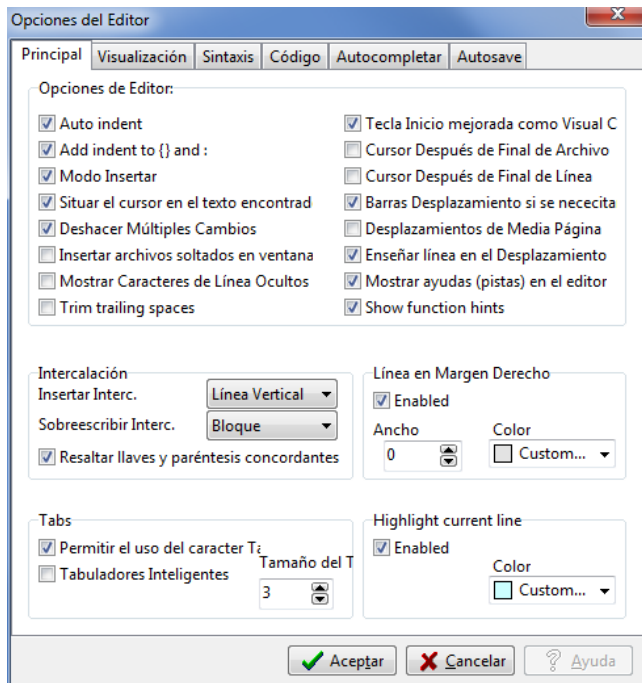
Instalación de Orwell Dev C++ en nuestra casa

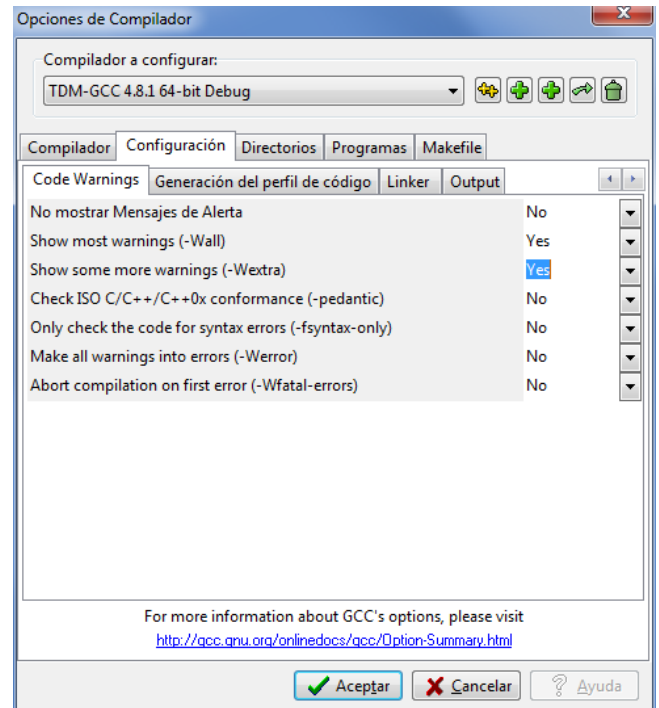
El entorno de desarrollo que usaremos será Orwell Dev C++. Puede descargarse desde la página:

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 varias 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.

- c) Usar `setlocale` (buscad documentación en Internet)

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

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

► **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 3) de este guión.

Actividad: Resolución de problemas.

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

6 (Interés bancario)

11 (Circunferencia)

12 (Gaussiana)

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

Realizad una lectura rápida de las actividades a realizar la semana próxima durante las horas de prácticas en las aulas de ordenadores (ver página siguiente)

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 las clases de prácticas en la segunda semana de clase.

El Entorno de Programación. Compilación de Programas

Arranque del Sistema Operativo

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

En la casilla etiquetada como Código, introduciremos fp. Al arrancar el SO, aparecerá una instalación básica de Windows con el compilador Orwell Dev C++. Todo lo que escribamos en la unidad C: se perderá al apagar el ordenador. Por ello, el alumno dispone de un directorio de trabajo en la unidad lógica U:, cuyos contenidos permanecerán durante todo el curso académico. En cualquier caso, es recomendable no saturar el espacio usado ya que, en caso contrario, el compilador podría no funcionar.

El alumno deberá crear el directorio U: \FP. 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 haced doble click sobre el fichero I_Pitagoras .cpp. Debe aparecer una ventana como la de la figura 1

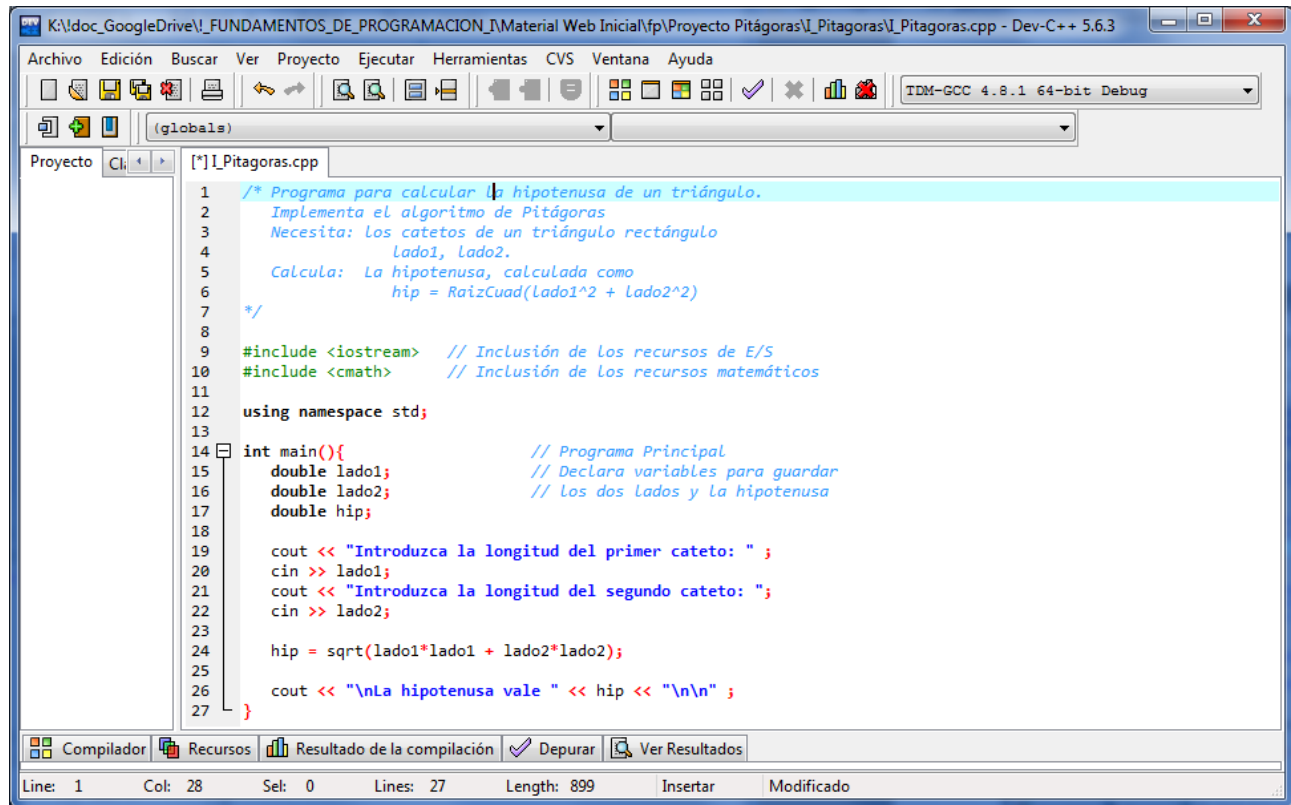


Figura 1: Programa que implementa el algoritmo de Pitágoras

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.

```
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");
}
```

Declaraciones

Entradas datos

Computos

Salida Resultados

Comentarios separados visualmente del código

líneas en blanco

espacios en blanco

Figura 2: Escritura de código

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

IMPORTANT

Compilación

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

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

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

```
Compilation succeeded
```

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

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 2 sobre la Ley de Ohm, de la relación de problemas I.

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

Archivo->Nuevo Código Fuente (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

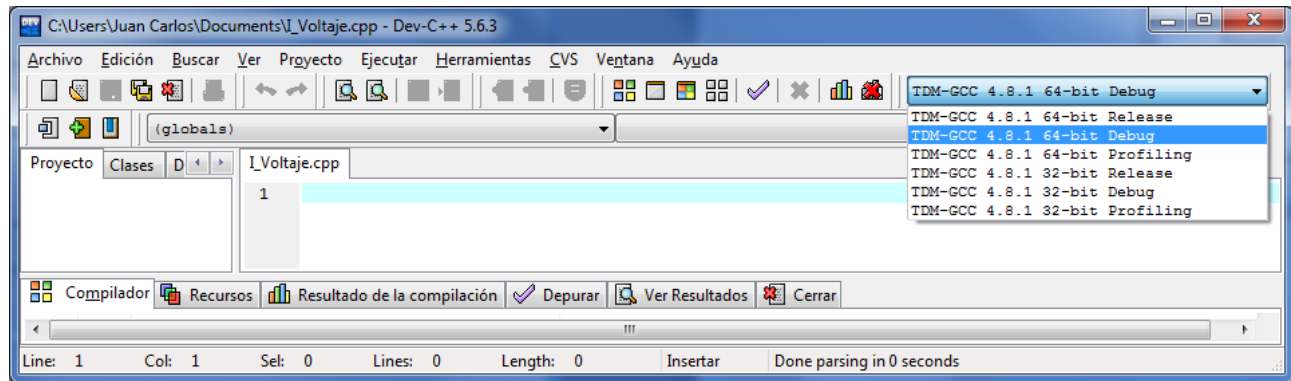


Figura 3: Creación de un programa nuevo

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.

Implementad los ejercicios que había que resolver en papel para esta sesión. Guardad los programas en el directorio en red. No hace falta entregar la solución.

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 domingo por la noche 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` (pero **no** los `.exe`).

- *Obligatorios:*
 - 5 (Población de China)
 - 16 (Segundos entre dos instantes)
 - 8 (Fabricante)
 - 9 (Intercambiar dos variables)
 - 10 (Media aritmética y desviación típica)
- *Opcionales:*
 - 14 (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, la siguiente web, para una introducción básica a los conceptos:

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

Si por ejemplo queremos ver las posibles combinaciones (con repetición e importando el orden) de dos elementos (0 y 1) en 4 posiciones de memoria (4 bits) obtenemos un total de $2^4 = 16$ posibilidades: 0000, 0001, 0010, ..., 1111. Ejecutad el siguiente applet para ver las combinaciones resultantes:

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



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

El profesor irá corrigiendo individualmente (a algunos alumnos elegidos aleatoriamente) los ejercicios indicados en la página anterior. Mientras tanto, el resto de alumnos deben intentar resolver los ejercicios siguientes de la Relación de Problemas I, página **RP-I.1**. Estos ejercicios no han de entregarse en `decsai`.

15 (Pasar a minutos y segundos enteros)

17 (Intercambiar tres valores)

Sesión 3

► **Actividades a realizar en casa**

Resolved los siguientes problemas:

- *Obligatorios:*

De la relación de Problemas I:

- 15 (Horas, minutos, segundos)
- 17 (Intercambiar tres variables)
- 20 (Pasar de mayúscula a minúscula)
- 21 (Pasar de carácter a entero)
- 23 (Expresiones diversas)
- 25 (Expresiones lógicas)

- *Opcionales:*

- 24 (Precisión y desbordamiento)

Actividades de Ampliación

Hojear la página

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

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



► Actividades a realizar en las aulas de ordenadores

Redireccionando las entradas de cin

Cada vez que se ejecuta `cin`, se lee un dato desde el periférico por defecto. Nosotros lo hemos hecho desde el teclado, introduciendo un dato y a continuación un ENTER. Otra forma alternativa es introducir un dato y luego un separador (uno o varios espacios en blanco o un tabulador). 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.

Resuelva los ejercicios siguientes de la Relación de Problemas I, página **RP-I.1**:

22 (Expresiones lógicas)

26 (Elección tipo de dato)

Estos ejercicios no han de entregarse en `decsai`.

Sesión 4

Estructura condicional

► **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 condicional

- **Obligatorios:**
 - 2 (Media)
 - 3 (Subir sueldo)
 - 5 (Tres valores ordenados)
 - 7 (Pasar de mayúscula a minúscula)
 - 8 (Pasar de mayúscula a minúscula y viceversa)
 - 9 (Descuento por ventas realizadas)
- **Opcionales:**
 - 13 (Renta bruta y neta)

► 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 2 (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.

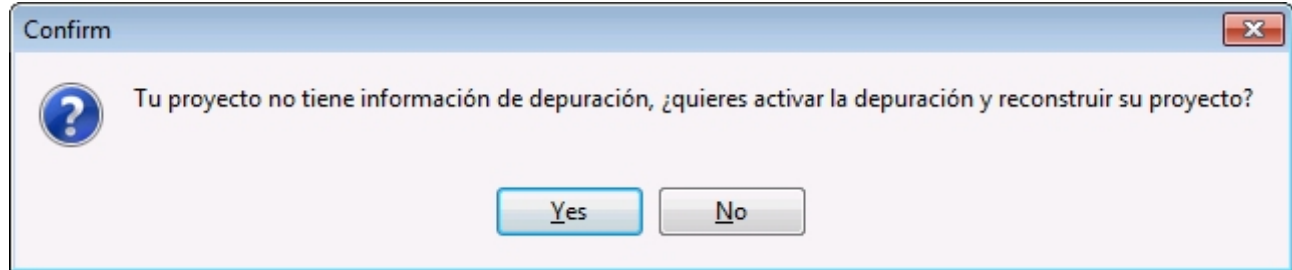


Figura 4: Ventana emergente que aparece cuando la configuración actual del compilador no permite tareas de depuración

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:

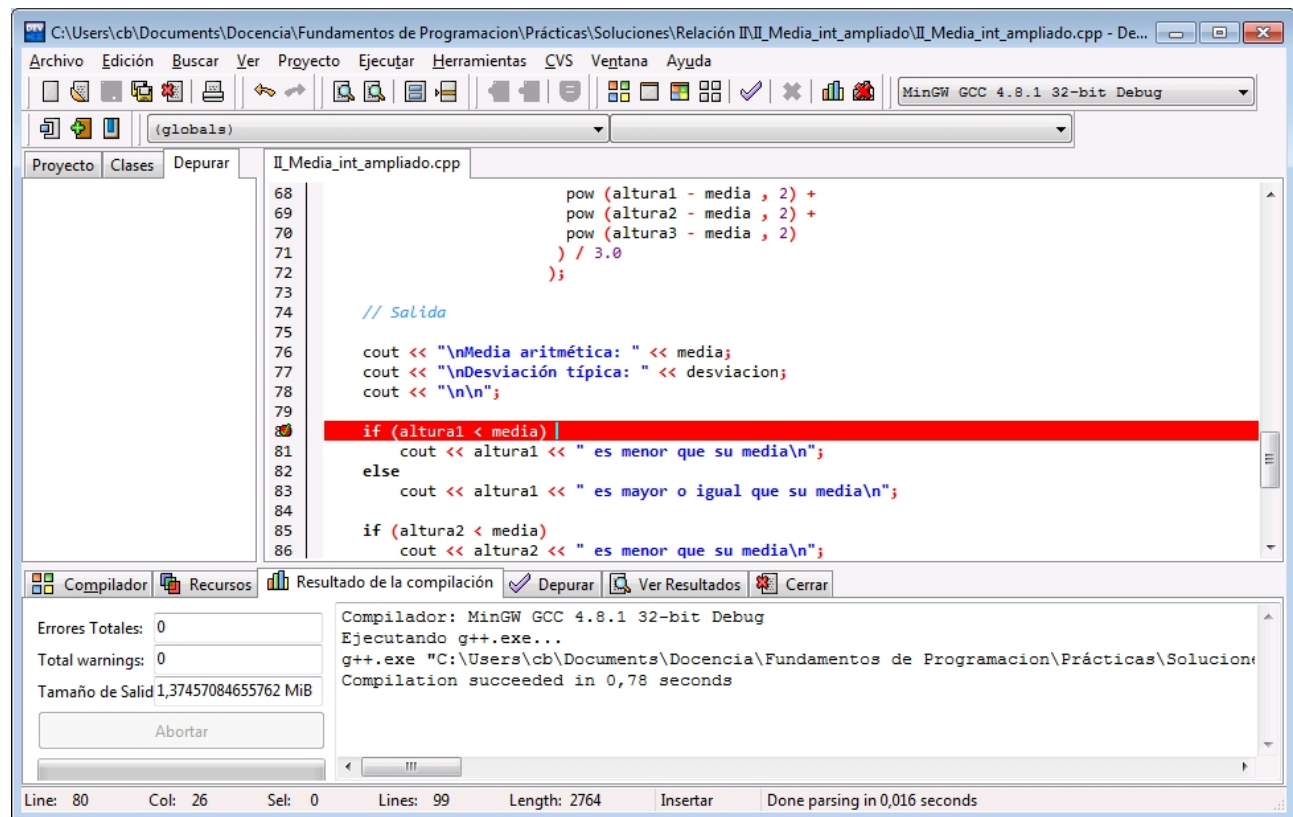



Figura 5: Se ha activado un punto de ruptura

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

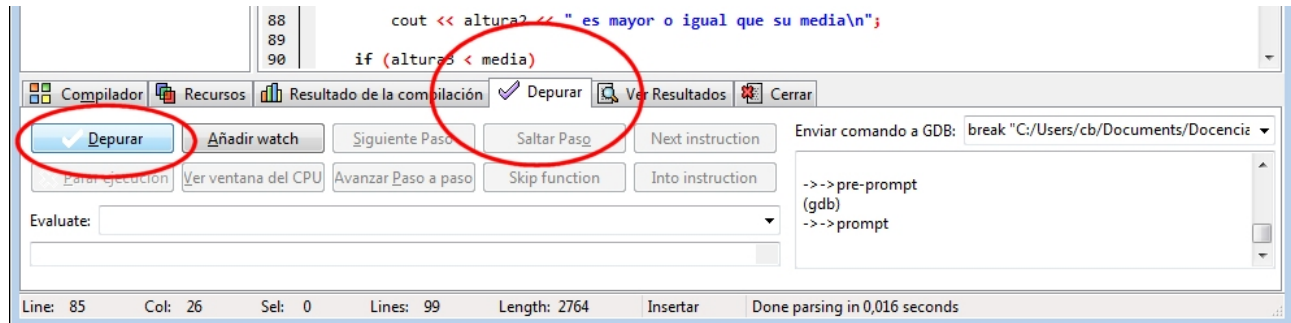


Figura 6: Inicio del proceso de depuración

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.

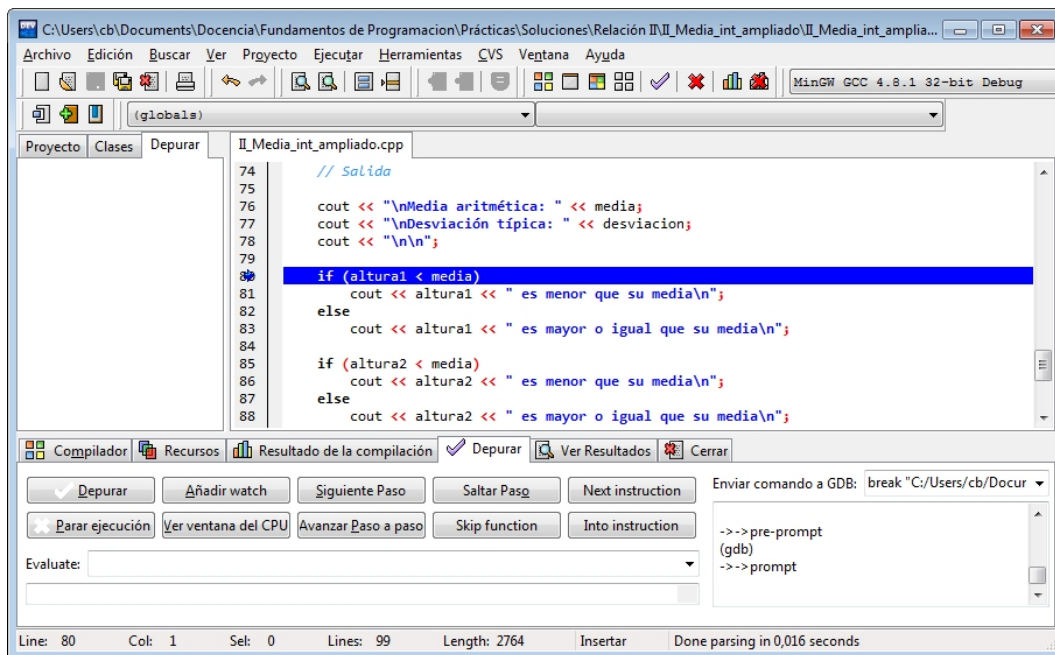


Figura 7: Inicio del proceso de depuración

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.

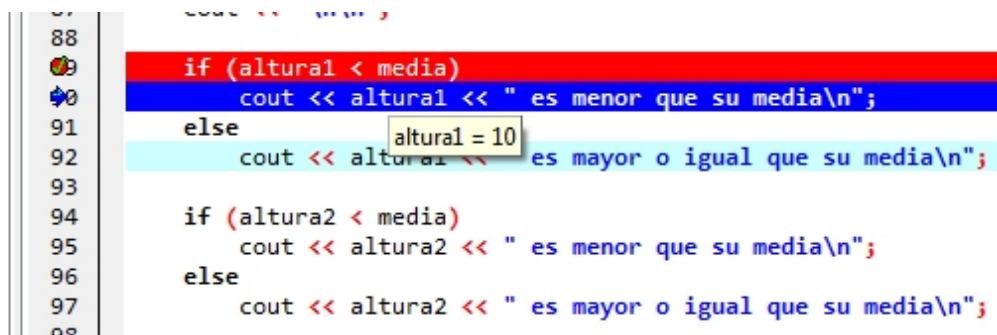


Figura 8: Inspeccionando el valor de una variable

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).

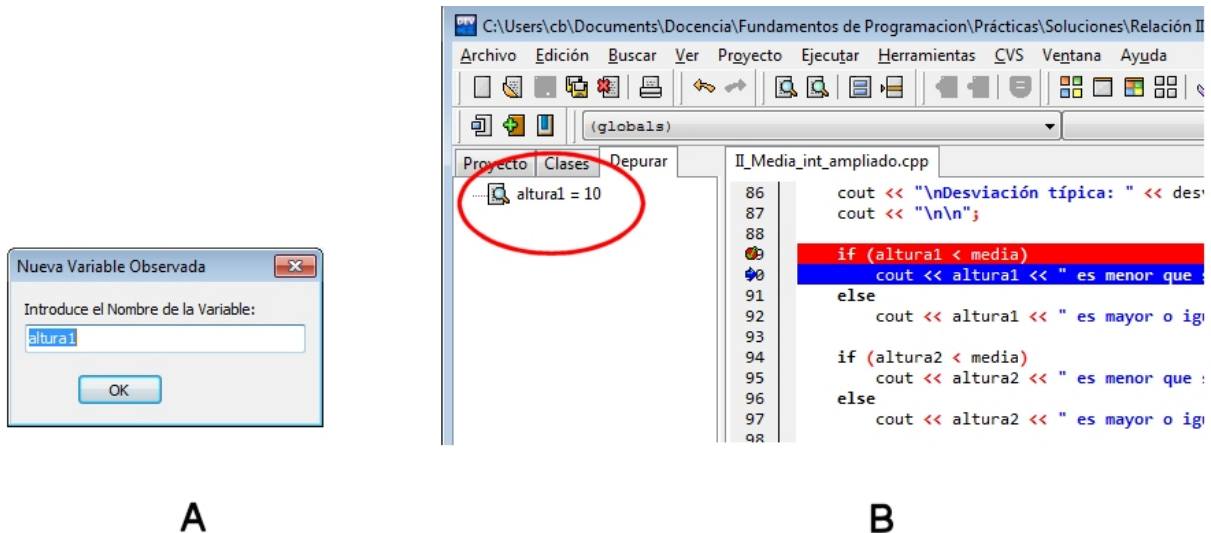


Figura 9: Añadiendo una variable para su inspección permanente

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".





Fundamentos de Programación.

Relaciones de Problemas.

RELACIÓN DE PROBLEMAS I. Introducción a C++

1. Indique cuáles serán los valores de las variables a y x después de ejecutar el código siguiente

```
a = 0;
i = 1;
x = 0;
a = a + i;
x = x + i / a;
a = a + i;
x = x + i / a;
a = a + i;
x = x + i / a;
a = a + i;
x = x + i / a;
```

Obsérvese que normalmente no usaremos nombres de variables tan cortos como los anteriores. Este ejemplo es una excepción, al tratarse de un ejercicio básico.

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

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

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

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

3. Escriba un programa que lea por pantalla la cantidad en millas (como un real) y muestre la cantidad equivalente en kilómetros. Debe tener en cuenta que 1 milla equivale a 1.609 kilómetros.

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

4. Realizar un programa que nos pida una longitud cualquiera dada en yardas. El programa deberá calcular el equivalente de dicha longitud en pulgadas, pies, millas y millas marinas, y mostrarnos los resultados en pantalla. Para el cálculo, utilice la siguiente tabla de conversión del sistema métrico:

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

1 pulgada= 25,4 milímetros
1 pie = 30,48 centímetros
1 yarda = 0,9144 metros
1 milla = 1609,344 metros
1 milla marina = 1852 metros

5. De <http://countrysmeters.info> se obtienen los siguientes datos estimados sobre la población de China:

- nace una persona cada 1.87 segundos
- muere una persona cada 3.27 segundos
- emigra una persona cada 71.9 segundos

Escriba un programa que muestre la población dentro de 2 años, considerando que la población actual es de 1.375.570.814 personas. Los datos de entrada son el número de años y la población de partida.

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

6. 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.

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

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

8. 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.

9. 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.

10. 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:

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

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

Estas medidas se utilizan mucho en Estadística para tener una idea de la distribución de datos. La media (mean en inglés) nos da una idea del valor central y

la desviación típica (standard deviation) nos da una idea de la dispersión de éstos. Ejecutad el programa con varios valores y comprobad que el resultado es correcto utilizando una calculadora científica o cualquier calculadora online como por ejemplo la disponible en <http://www.disfrutalasmaticas.com/datos/desviacion-estandar-calculadora.html>

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

11. 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, recompilad y ejecutad (La parte de compilación y ejecución se realizará cuando se vea en clase de prácticas el entorno de programación).

¿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.

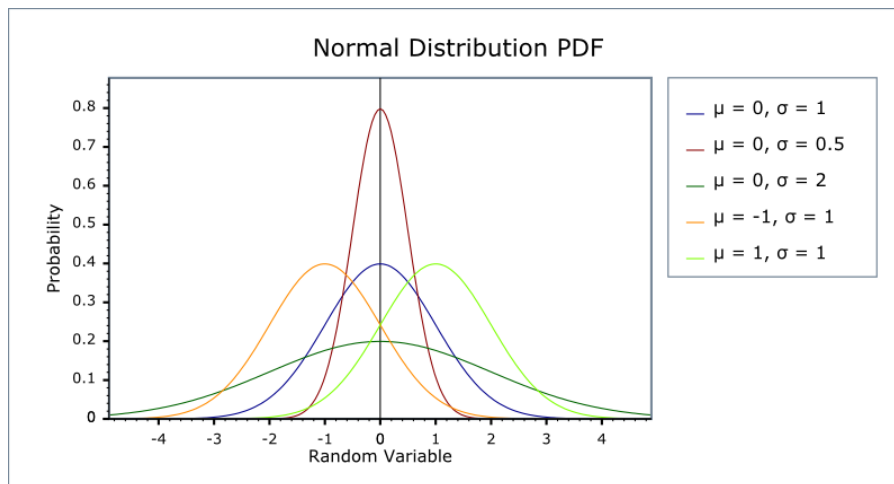
Cambiad ahora el valor de la constante PI por el de 3.1415927, recompilad y ejecutad.

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

12. 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

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

La función gaussiana es muy importante en Estadística. Es una función real de variable real en la que el parámetro μ se conoce como *esperanza* o *media* y σ como *desviación típica* (*mean* y *standard deviation* en inglés). En la gráfica de abajo pueden verse algunos ejemplos de esta función con distintos parámetros.



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 cualquiera de las calculadoras disponibles en:

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

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

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

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

Escribid dos programas para convertir entre estas dos medidas:

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

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

14. 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.

15. Leer 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.

16. Calcular el número de segundos que hay entre dos instantes del mismo día.

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

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

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

17. 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.

18. 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.

19. 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.

20. 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.

21. Supongamos el siguiente código:

```
int entero;
char caracter;

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

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.

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

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

23. 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.

24. 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.

25. 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.

26. 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.

27. 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.

28. 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.

29. 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`.

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

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

Algunas funciones de `cmath`

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

$\text{cos}(x) \rightarrow \cos(x)$

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

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

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

Dificultad Baja.

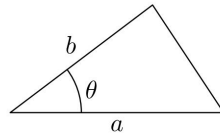
31. Dos locomotoras parten de puntos distintos avanzando en dirección contraria sobre la misma vía. Se pide redactar un programa para conocer las distancias que habrán recorrido ambas locomotoras antes de que choquen teniendo en cuenta que la primera locomotora viaja a una velocidad constante V_1 , que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

32. 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.

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

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

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

Dificultad Media.

RELACIÓN DE PROBLEMAS II. Estructuras de Control

Ejercicios sobre condicionales

1. (Parking Madrid)

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

Desde el minuto 0 al 30: 0.0412 euros cada minuto
Desde el minuto 31 al 90: 0.0370 euros cada minuto
Desde el minuto 91 al 660: 0.0493 euros cada minuto
Desde el minuto 661 hasta máximo 24 horas: 31.55 euros

Dado un tiempo de entrada y un tiempo de salida, construya un programa que calcule la tarifa final en euros a cobrar. Ejemplo: si el tiempo de permanencia es de 32 minutos, los primeros 30 minutos se facturan a 0.0412 el minuto y los 2 restantes a 0.0370.

Finalidad: Utilización del condicional simple. Dificultad Baja.

2. Ampliad el ejercicio 10 de la relación de problemas I, para que, una vez calculada la media y la desviación, el programa imprima por cada uno de los valores introducidos previamente, si está por encima o por debajo de la media. Por ejemplo:

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

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

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

3. 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.

¿Es mejor incluir otra variable nueva `salario_final` o es mejor modificar la variable que teníamos?

Imprimid el resultado por pantalla. En caso contrario imprimid el mensaje "No es aplicable la subida". En ambos casos imprimid el salario resultante.

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

RELACIÓN DE PROBLEMAS II. Estructuras de Control

4. 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.

5. 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.

6. 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.

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

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

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

8. Queremos modificar el ejercicio 7 para leer un carácter `letra_original` desde teclado y hacer lo siguiente:

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

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

Finalidad: Plantear una estructura condicional anidada. Dificultad Baja.

9. En clase de teoría se ha visto e implementado el siguiente ejemplo:

En un programa de ventas, si la cantidad vendida es mayor de 100 unidades, se le aplica un descuento del 3 %. Por otra parte, si el precio final de la venta es mayor de 700 euros, se aplica un descuento del 2 %. Ambos descuentos son acumulables.

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

Cambiar el programa visto en clase para incorporar este nuevo criterio.

Finalidad: Plantear una estructura condicional anidada. Dificultad Baja.

10. 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 4% si es mayor de 65 o menor de 35 años. Si además de cumplir la anterior condición, también tiene un salario inferior a 300 euros, se le subirá otro 3%.

Imprimid el resultado por pantalla.

Finalidad: Plantear una estructura condicional anidada. Dificultad Baja.

11. Modificad las soluciones de los ejercicios 3,4,5,6 para que no se mezclen E/S y C (entradas/salidas y cálculos) dentro de la misma estructura condicional.

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

12. Modificad la solución al ejercicio 8 para que, dependiendo de cómo era la letra introducida, imprima en pantalla alguno de los siguientes mensajes:

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

Hágalo separando E/S y C.

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

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

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

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

Los datos a leer son:

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

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

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

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

Finalidad: Plantear una estructura condicional anidada. Dificultad Media.

14. Modificad el ejercicio 5 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.

15. Para resolver el ejercicio 12 se podrían usar dos variables lógicas (`es_mayuscula`, `es_minuscula`), que nos proporcionen 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 8 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.

16. Una compañía aérea establece el precio del billete como sigue: en primer lugar se fija una tarifa base de 150 euros, la misma para todos los destinos. Si el destino está a menos de 200 kilómetros, el precio final es la tarifa inicial. Para destinos a más de 200 Km, se suman 10 céntimos por cada kilómetro de distancia al destino (a partir del Km 200). En una campaña de promoción se va a realizar una rebaja lineal de 15 euros a todos los viajes. Además, se pretenden añadir otras rebajas y se barajan las siguientes alternativas de políticas de descuento:

- a) Una rebaja del 3 % en el precio final, para destinos a más de 600Km.
- b) Una rebaja del 4 % en el precio final, para destinos a más de 1100Km. En este caso, no se aplica el anterior descuento.
- c) Una rebaja del 5 % si el comprador es cliente previo de la empresa.

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.

17. Vamos a modificar el ejercicio 4 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, resolved 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. ¿Cual sería la mejor? Escoged una e implementar 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 quién es el dividendo y quién el divisor:

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

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.

Ejercicios sobre bucles

18. 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.

19. Modifiquemos el ejercicio 6 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.

20. 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.

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

21. 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 ctype.

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

22. 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

RELACIÓN DE PROBLEMAS II. Estructuras de Control

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.

23. Ampliad el ejercicio 5 de la relación de problemas I.

Esta nueva versión del programa pedirá un valor de población y calculará cuántos años (enteros) transcurrirán hasta que la población estimada sea mayor o igual al valor dado.

Nota: Filtrar el valor de población introducido de manera que sea mayor que el de la población actual.

Finalidad: Practicar con filtros y ciclos básicos. Reutilizar código ya escrito y verificado. Dificultad Baja.

24. Ampliad el ejercicio 16 de manera que se permita que los dos instantes puedan pertenecer a dos días distintos, pero eso sí, consecutivos.

Filtrar adecuadamente los datos leídos.

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

25. Ampliad el ejercicio 6. El programa pedirá los valores de dos años y mostrará todos los años bisiestos comprendidos entre los dos valores dados.

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

26. 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.

27. En el ejercicio 14 de la Relación de Problemas I se pedía escribir un programa que leyese un valor entero de tres dígitos e imprimiese los dígitos separados por un espacio en blanco. 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.

28. Escribir un programa que lea dos números enteros, escriba en pantalla el menú que se muestra a continuación, lea la opción seleccionada y muestre el resultado de la operación indicada.

```
-----  
I.  Introducir números  
S.  Suma  
P.  Producto  
M.  Máximo  
R.  Salir  
-----
```

Introduzca opción:

Observe que se permite realizar distintas operaciones (opciones S,P,M) con los mismos valores numéricos (los introducidos en la opción I)

Finalidad: Ejercitar el uso de bucles, junto con otras estructuras de control. Dificultad Baja.

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

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

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

```
#include <iostream>  
#include <ctime>  
#include <cstdlib>  
using namespace std;  
  
int main(){  
    const int MIN = 1, MAX = 100;  
    const NUM_VALORES = MAX-MIN + 1;           // rango  
    int incognita;                               // número generado  
    time_t tiempo;  
  
    // Inicialización de la secuencia:  
    srand(time(&tiempo));  
  
    // Generación de un número aleatorio incognita:  
    // MIN <= incognita <= MAX  
    incognita = (rand() % NUM_VALORES) + MIN;
```

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

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

Dificultad Media.

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

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

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

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

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

Finalidad: Ejercitar los bucles. Dificultad Baja.

32. 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.

33. 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.

34. 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.

35. 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 18 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 19 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.

36. Supongamos una serie numérica cuyo término general es:

$$a_i = a_1 r^{i-1} \longrightarrow a_1, a_1 r, a_1 r^2, a_1 r^3, \dots$$

RELACIÓN DE PROBLEMAS II. Estructuras de Control

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

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

Se proponen dos alternativas:

- a) 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`.
- b) Si analizamos la expresión algebraica de la serie numérica, nos damos cuenta que es una *progresión geométrica* ya que cada término de la serie queda definido por la siguiente expresión:

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

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

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.

37. Reescribid la solución a los ejercicios 18 (divisores) y 19 (interés) usando un bucle `for`

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

38. 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.

39. 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

RELACIÓN DE PROBLEMAS II. Estructuras de Control

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 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.

40. Sobre la solución del ejercicio 19 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.

41. 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 un elemento del conjunto {min_izda ... max_izda} y otro elemento del conjunto {min_dcha ... max_dcha}. Por ejemplo, si min_izda = b, max_izda = d, min_dcha = j, max_dcha = m, el programa debe imprimir las parejas que pueden formarse con un elemento de {b c d} y otro elemento de {j k l m}, es decir:

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

Finalidad: Ejercitar los bucles anidados. Dificultad Baja.

42. ([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.

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

```
*****
*****
*****
```

Finalidad: Ejercitar los bucles anidados. Dificultad Baja.

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

```
  *
 ***
*****
*****
 ***
 ***
```

Finalidad: Ejercitar los bucles anidados. Dificultad Media.

45. 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
```

RELACIÓN DE PROBLEMAS II. Estructuras de Control

```
4 5 6
5 6
6
```

Finalidad: Ejercitar los bucles anidados. Dificultad Baja.

46. 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.

47. 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.

48. 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.

49. Recuperad la solución del ejercicio 12 (función gaussiana) de la relación de problemas I. Se pide crear un menú principal para que el usuario pueda elegir las siguientes opciones:

Introducir (P)arámetros de la función (esperanza y desviación)
(S)alir del programa

Si el usuario elige la opción de salir, el programa terminará; si elige la opción de introducir los parámetros, el programa leerá los dos parámetros (esperanza y desviación) y a continuación se le presentará un menú con las siguientes opciones:

Introducir varios (V)alores de abscisas
Volver al menú (A)nterior

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

Finalidad: Ejercitar los bucles anidados. Dificultad Media.

50. 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.

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

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

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

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

Dificultad Baja.

52. 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.

53. 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.

RELACIÓN DE PROBLEMAS II. Estructuras de Control

54. 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.

55. 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.

56. 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.

57. 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.

58. Diremos que un número entero positivo es secuenciable si se puede generar como suma de números consecutivos. Por ejemplo, $6 = 1 + 2 + 3$, $15 = 7 + 8$. Esta descomposición no tiene por qué ser única. Por ejemplo, $15 = 7 + 8 = 4 + 5 + 6 =$

RELACIÓN DE PROBLEMAS II. Estructuras de Control

1 + 2 + 3 + 4 + 5. Escribir un programa que lea un entero n y nos diga cuántas descomposiciones posibles tiene. Por ejemplo:

15 -> 3 descomposiciones
94 -> 1 descomposición
108 -> 3 descomposiciones

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

Dificultad Media.

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

Dificultad Media.

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

Dificultad Baja.

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

Dificultad Baja.

62. 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.

63. 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

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

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

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

Dificultad Media.

64. 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.

65. 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.

66. 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.

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

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

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

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

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

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

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

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

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

Se pueden construir aproximaciones al número áureo mediante la fórmula $a_n = \frac{f_{n+1}}{f_n}$ siendo f_n el número de Fibonacci de orden n (ver problema 67).

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

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

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

Dificultad Media.

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

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

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

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

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

RELACIÓN DE PROBLEMAS II. Estructuras de Control

sucesión infinita de período 1. Un *número amigable* produce una sucesión infinita de período 2 (como el 220 ó 284).

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

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