UNIVERSIDAD DE GRANADA

DECSAI

Departamento de Ciencias de la Computación e Inteligencia Artificial

Fundamentos de Programación

Doble Grado en Ingenieria Informática y Matemáticas

Francisco José Cortijo Bon

cb@decsai.ugr.es

Guión de Prácticas

Curso 2020 - 2021

Departamento de Ciencias de la Computación e Inteligencia Artificial

Universidad de Granada

"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".

Fundamentos de Programación

Guión de Prácticas

Curso 2020/2021

Autor: **Francisco José Cortijo Bon** (cb@decsai.ugr.es)

Basados en el guión original de: **Juan Carlos Cubero** (JC.Cubero@decsai.ugr.es)
Departamento de Ciencias de la Computación e Inteligencia Artificial
Universidad de Granada.

Índice del Guión de Prácticas

Sobre el guión de prácticas

Este guión de prácticas contiene las actividades a realizar por el alumno. Gran parte de ellas las hará de manera autónoma. La resolución de los ejercicios de programación propuestos es una tarea fundamental para superar con éxito la asignatura ya que la programación es una tarea que requiere de un trabajo continuado, organizado y ordenado. Las habilidades, técnicas y destrezas que se requieren en la programación no se pueden adquirir leyendo o estudiando simplemente, sino que necesitan de un trabajo práctico continuado.

El guión está dividido en **sesiones**, que corresponden a cada semana lectiva. Las actividades no presenciales (las encontrará baja el epígrafe *Actividades a realizar en casa*) de cada sesión deben realizarse antes de la correspondiente sesión de prácticas. Cada sesión se desarrolla en un periodo de 2 horas en la que el profesor interviene e interactúa de manera síncrona a la participación de los alumnos. Las sesiones están organizadas para, en base a los contenidos desarrollados en las clases de teoría, proporcionar un contenido eminentemente práctico. Así, el profesor puede explicar el entorno de programación, corregir los ejercicios propuestos, ampliar profundizar sobre contenidos teóricos sobre problemas concretos, etc.

La primera sesión presencial de prácticas (página 9) tendrá lugar durante la segunda semana de clases, por lo que el alumno deberá realizar con anterioridad las actividades encargadas.

Una actividad autónoma recurrente es la resolución de problemas, organizados en **Relaciones de Problemas**, que encontrará al final de este documento.

La organización de cada una de las sesiones de prácticas será, generalmente, la siguiente.

- El profesor propone con antelación suficiente la resolución de una serie de ejercicios de una relación de problemas (enumeradas en el epígrafe **Actividades a realizar en casa**). Estos ejercicios deberán resolverse en casa, y las soluciones se entregarán de acuerdo las indicaciones y plazos establecidos usando la plataforma **PRADO**.
- Los ejercicios más importantes serán corregidos en las sesiones de prácticas. Los alumnos deberán defenderlos individualmente (públicamente, explicándolo a sus compañeros). La calidad del trabajo y la defensa realizada forma parte de la nota final de la asignatura (10 % tal y como se explica en el método de evaluación de la asignatura).
- El profesor propondrá soluciones alternativas/mejoras a toda la clase y se debatirá sobre ellas, y en su caso, serán implementadas en la sesión activa para ser evaluadas.

Del conjunto de problemas que forman las relaciones de problemas seleccionamos para cada sesión una serie de problemas obligatorios y opcionales.

1. *Obligatorios*: Como su nombre indica hay que realizarlos y entregarlos para que la entrega sea considerada para su evaluación.

2. *Opcionales*: Son ejercicios que complementan (generalmente) ejercicios obligatorios y pueden ser una buena motivación para ampliar la funcionalidad de éstos. El alumno debería intentar resolver por su cuenta un alto porcentaje de éstos.

Para la realización de estas prácticas, se utilizará, preferentemente, 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 y usar en su casa cualquier otro compilador, como por ejemplo `Code::Blocks` → `http://www.codeblocks.org/`

Es muy importante que la asignatura se lleve al día para poder realizar los ejercicios propuestos .

Instalación de Orwell Dev C++ en nuestra casa

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

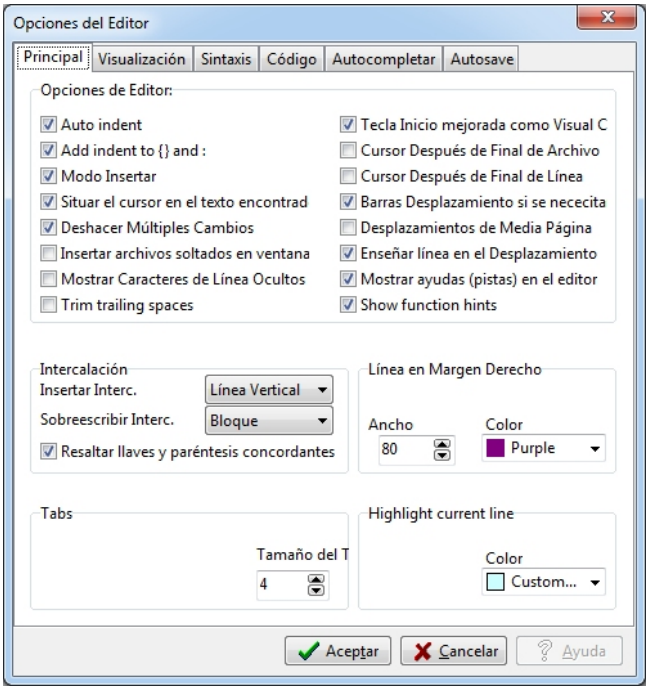
http://sourceforge.net/projects/orwelldevcpp/

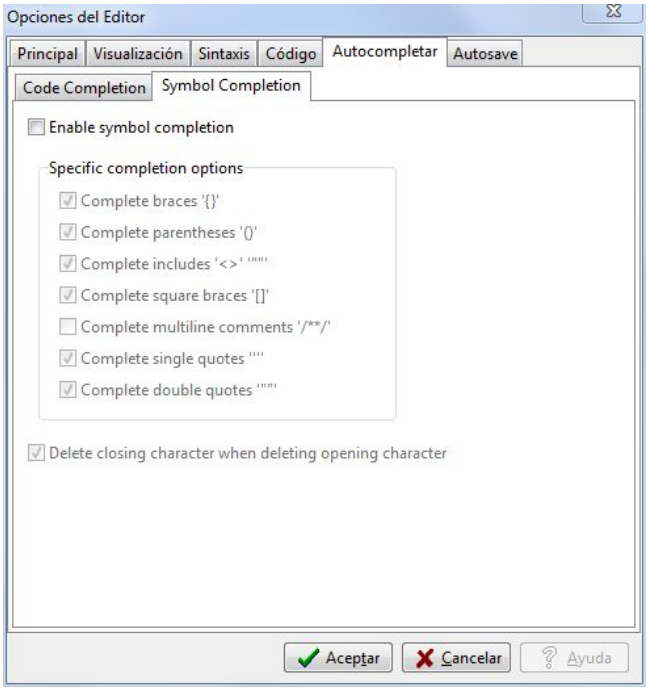
En este momento están disponibles en la misma descarga descarga las versiones 5.11 del entorno de trabajo y 4.9.2 del compilador:

- Dev-Cpp 5.11 (http://orwelldevcpp.blogspot.com.es/)
- TDM-GCC 4.9.2 (https://jmeubank.github.io/tdm-gcc/)

Cuando lo instalemos en nuestra casa, configurar las siguientes opciones del **editor** (en el menú principal: `Herramientas` | `Opciones del Editor`) :

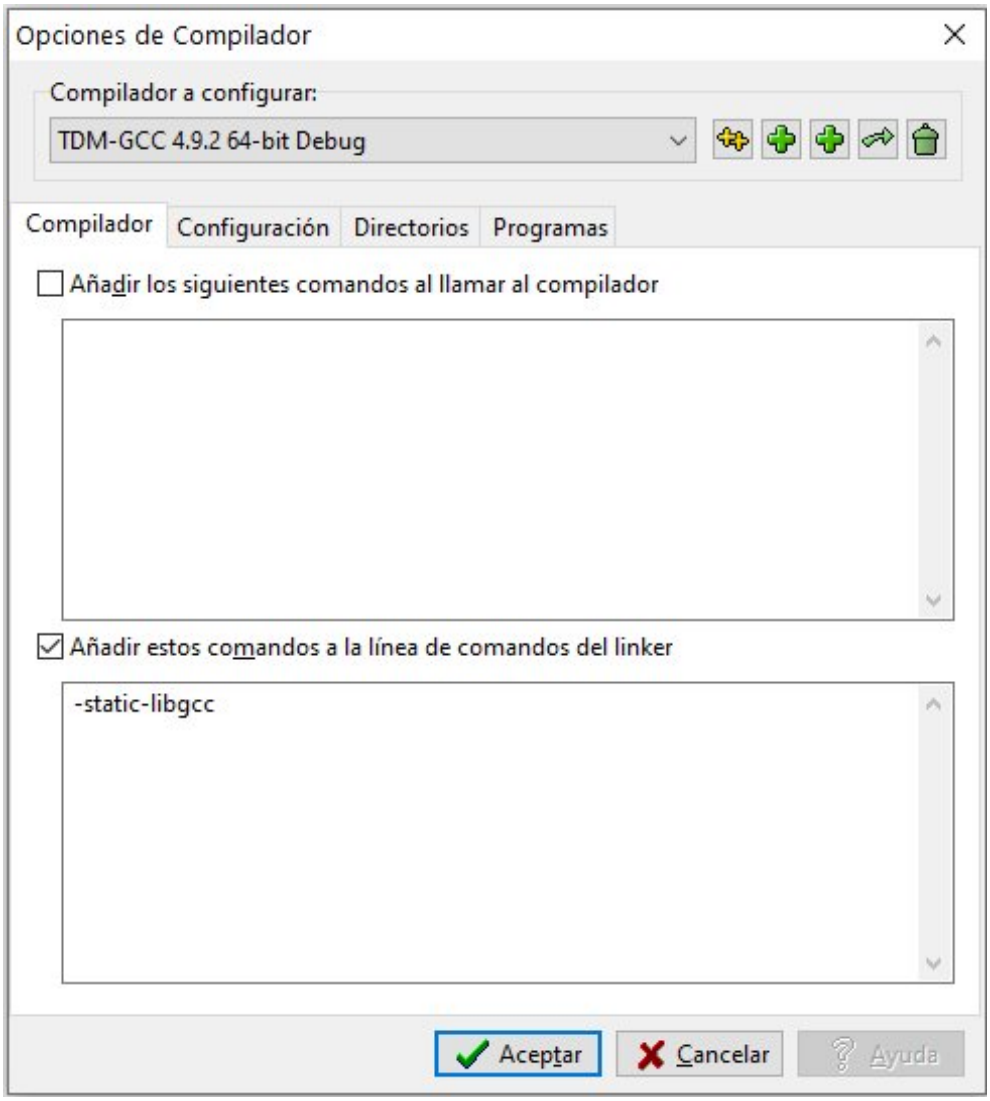
```
Herramientas -> Opciones del editor
   -> Principal
      Desmarcar Tabuladores inteligentes
      Tamaño del tabulador: 4
      Línea en Margen Derecho --> Ancho: 80  Color: Purple
   -> Autocompletar -> Symbol completion
      Desmarcar Enable Symbol completion
```





Después, configurar las siguientes opciones del **compilador** (en el menú principal: `Herramientas` | `Opciones del Compilador`).

En primer lugar indicaremos que el compilador a configurar será el TDM-GCC 4.9.2 y usaremos la versión de 64 bits con el perfil de trabajo `Debug` (*depuración*):

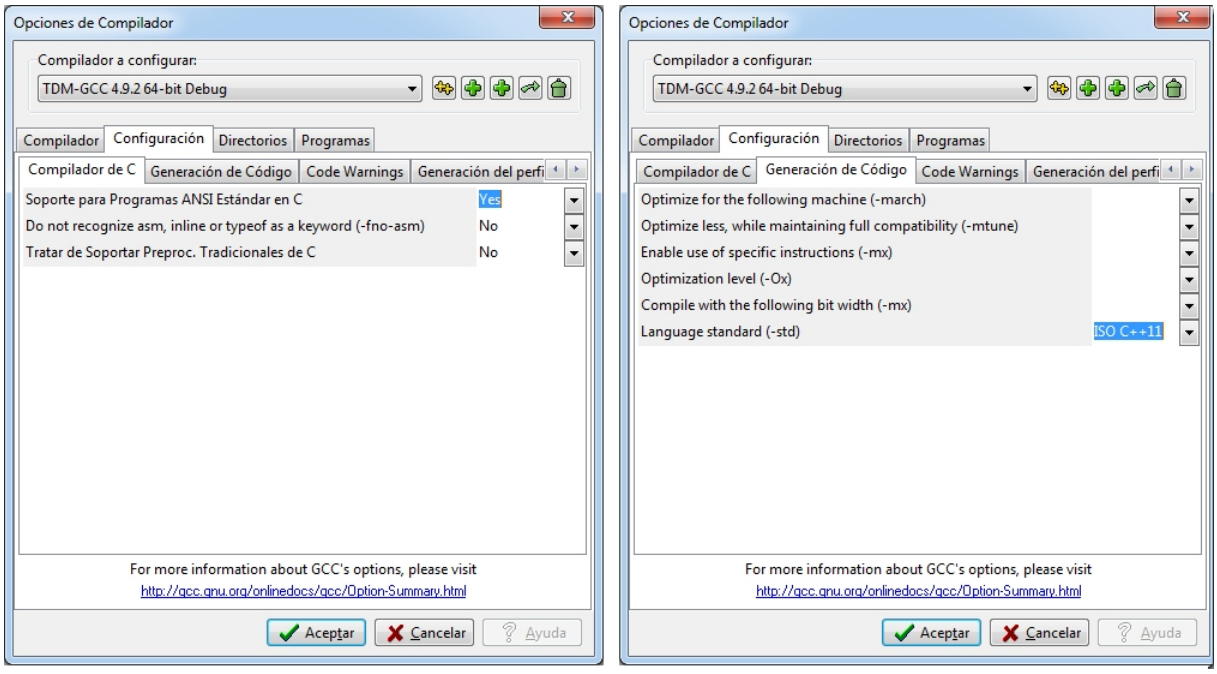


Debemos asegurarnos que se va a compilar correctamente código que sigue los estándares de programación para los lenguajes C y C++:

- ANSI C, en el caso de programar en C y
- ISO C++ 11, para C++.

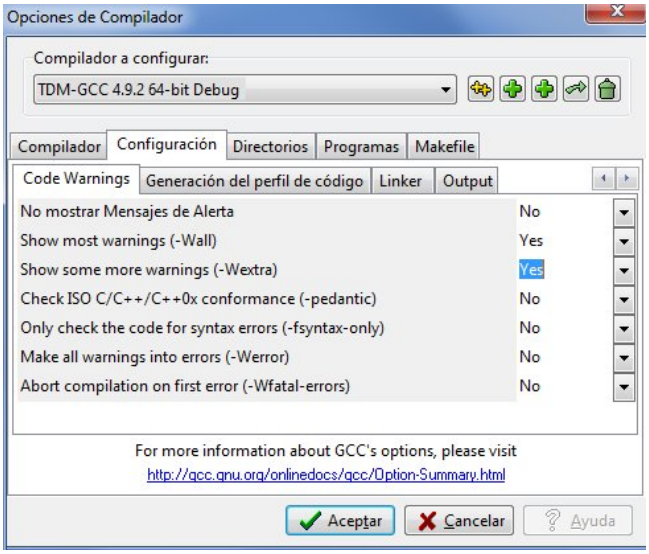
  Puede leer en Wikipedia (https://es.wikipedia.org/wiki/C%2B%2B#C++11)
  En 2011 C++11 inauguró una nueva era en la historia de C++, iniciando un ciclo trienal de lanzamiento de nuevas versiones. A C++11 le siguió C++14 y luego C++17, que es la versión actual en 2019; C++20 se encuentra próximo a estandarizarse, y ya se está trabajando en la versión C++23. Los compiladores intentan adelantarse incorporando de manera experimental algunas novedades antes de los lanzamientos oficiales. Pero cada nueva versión de C++ incluye tal cantidad de agregados que los compiladores más adelantados no suelen terminar de incorporarlos hasta dos o tres años después del lanzamiento de esa versión.

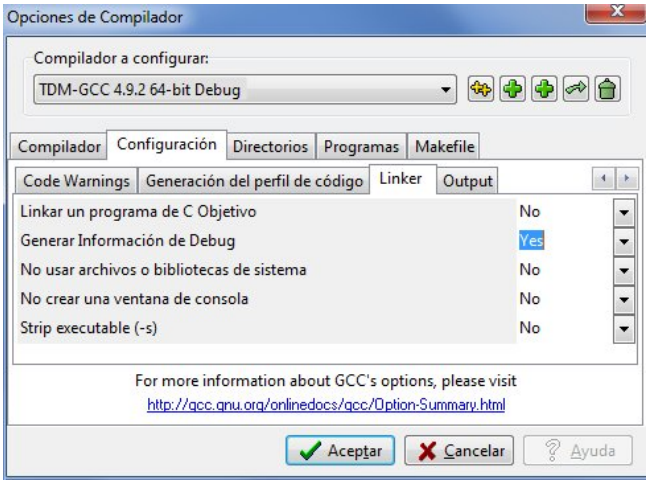
```
Herramientas -> Opciones del Compilador ->
    Configuración -> Compilador de C
        Soporte para Programas ANSI Estándar en C: Yes

Herramientas -> Opciones del Compilador ->
    Configuración -> Generación de Código
        Language standard (-std): ISO C++ 11
```



Después se configurará qué tipo de avisos (*warnings*) queremos que se activen para el proceso de compilación (izquierda) e indicaremos que queremos generar información de depuración (*bebug*) para poder monitorizar la ejecución del programa (derecha):

```
Herramientas -> Opciones del Compilador ->
    Configuración -> Code Warnings
        Show most warnings: Yes
        Show some more warnings: Yes

Herramientas -> Opciones del Compilador ->
    Configuración -> Linker
        Generar información de Debug: Yes
```





Aunque no es necesario un conocimiento detallado, debe saber que la opción:

```
-finput-charset=cp1252 -fexec-charset=cp850
```

indica al compilador que el código fuente está codificado con la página de códigos 1252 (de Windows) y que el ejecutable (la consola) será compatible con la 850 (ver apuntes de la asignatura).

De esta forma conseguimos mostrar los acentos y los caracteres especiales como la **ñ** en la consola que se abre al ejecutar nuestros programas.

Por otra parte, la opción

```
-Wl,--stack,21000000
```

indica al enlazador (*linker*) que utilice un tamaño de memoria para la pila de 21000000 bytes. Esto será necesario cuando usemos vectores con tamaños muy grandes (tema IV).

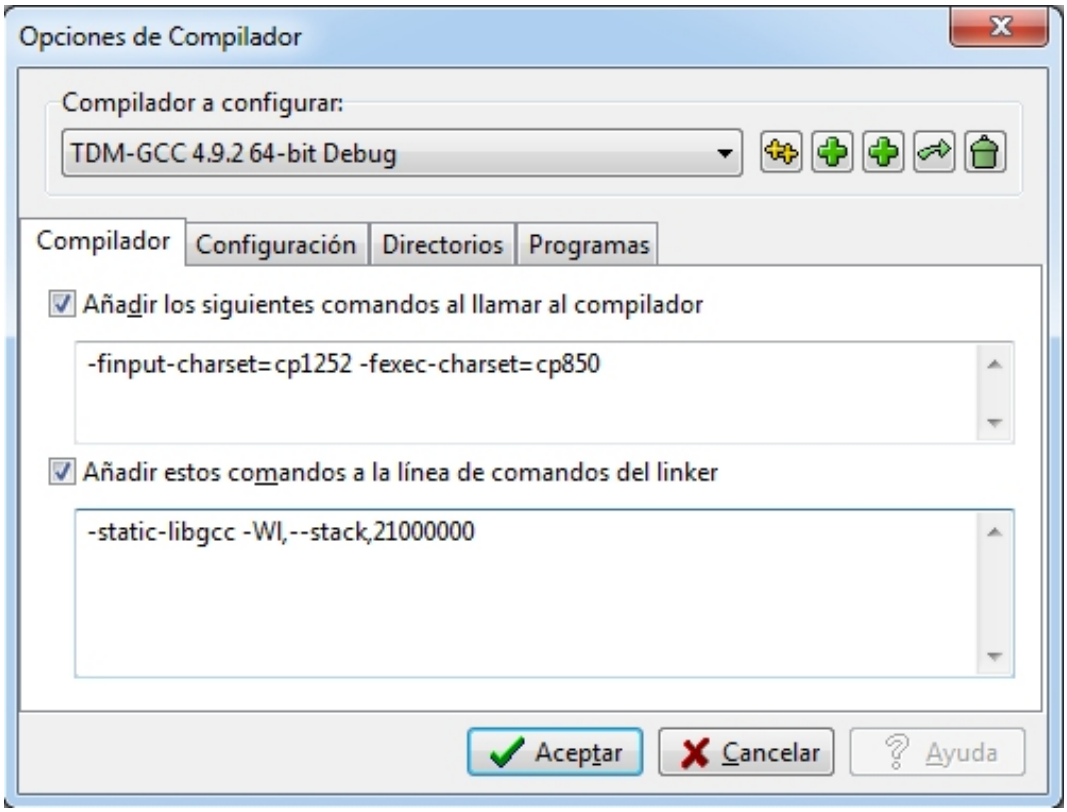


Tabla resumen de accesos directos usados en Orwell Dev C++

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
Shift F7	Avanzar paso a paso
	Ejecución paso a paso entrando en los métodos o funciones
Ctrl F7	Saltar paso
	Ejecución hasta el siguiente punto de ruptura
Shift F4	Ir a cursor
	Ejecuta sentencias hasta llegar a la posición actual del cursor

Sesión 1

El Entorno de Programación. Compilación de Programas

La primera sesión de prácticas se desarrolla en un periodo de tiempo en el que el alumno desconoce los aspectos más básicos de la programación. En las clases de teoría está familiariazándose con los tipos de datos básicos, expresiones, y entendiendo el mecanismo de entrada/salida.

Es posible, no obstante, empezar a programar usando un **modelo basado en ejemplos**. A partir de programas sencillos que el profesor le proporciona pueden proponerse otros problemas que se resuelven con una estrategia similar y que permiten investigar y aprender de manera autónoma nuevas funciones de la biblioteca estándar de C++. Los primeros ejercicios que se pueden resolver están basados en el modelo simple de tres etapas:

1. **Entrada**: lectura de los valores de los datos que intervienen en el programa.
2. **Cálculos**: obtener los valores de los datos en los que estamos interesados a partir de los datos de entrada (normalmente traduciendo una o más fórmulas/ecuaciones matemáticas a expresiones y asignaciones en C++).
3. **Salida**: presentación de los resultados obtenidos.

Para poder realizar un trabajo práctico real se requiere escribir el programa en un **editor** de texto y usar un **compilador** para generar el programa ejecutable. El proceso de compilación y generación de código ejecutable puede ser una tarea larga y compleja, especialmente si se trabaja en proyectos complejos en el que intervienen diferentes equipos de trabajo que pueden trabajar con diferentes herramientas de desarrollo. En esta asignatura, no obstante, trabajaremos en proyectos simples y emplearemos herramientas simples (se trata, no lo olvide, del primer contacto con la programación de ordenadores). Durante esta primera práctica podrá escribir y compilar sus programas con relativamente poco esfuerzo. Emplearemos un sencillo **IDE** (**I**ntegrated **D**evelopment **E**nvironment, ó entorno de desarrollo integrado) con este objetivo.

Como requisito previo el alumno debe haber resuelto sus obligaciones previas relativas a la obtención de las credenciales (cuenta y correo en la UGR) que le identifican como alumno de la UGR. No olvide que necesita disponer también de las credenciales para el acceso a los ordenadores de las aulas de prácticas de la ETSIIT. No hace falta decir que es vital disponer de un ordenador para el estudio y la realización de las tareas de prácticas.

► Objetivos

Los objetivos de esta sesión de prácticas son:

1. Descargar e instalar en su ordenador personal el entorno de desarrollo **Dev C++**.
2. Configurar las características básicas de Dev C++.
3. Ser capaces de cargar un fichero `.cpp` en Dev C++ y grabar un fichero `.cpp`
4. Compilar un programa en C++ y entender qué es y qué consecuencias tiene.
5. Ejecutar un programa escrito en C++.
6. Leer, entender y actuar ante errores de compilación y ejecución.
7. Detectar (diseñando casos de prueba) los errores lógicos en los programas y actuar ante ellos.
8. Crear nuevos programas en C++.

► Actividades a realizar en casa

Actividad: Conseguir login y password.

Antes de empezar la primera sesión de prácticas el alumno debe disponer de una cuenta en la Universidad, tal y como se indica en
http://csirc.ugr.es/informatica/cuentasinstitucionales/AccesoIdentificado/index.html

> *El Portal Web de Acceso Identificado es un servicio personalizado de consulta y tramitación web de la Universidad de Granada que le da acceso a gestiones, servicios y aplicaciones informáticas (desarrolladas por el CSIRC para UGR) tales como ayudas del GAS, consulta de expedientes académicos, actas académicas, nóminas, ... entre otros muchos, y que se muestran o no dependiendo del tipo de usuario que entra: alumno, PDI o PAS.*
> *Por ahora, la cuenta de Acceso Identificado no caduca, aunque su relación con la Universidad finalice; la cuenta seguirá estando activa pero el acceso a las distintas aplicaciones que contiene el portal web sí puede estar restringido, según su perfil, vínculo o situación en la Universidad.*

Para obtener una dirección de correo electrónico de la Universidad, hay que seguir las instrucciones detalladas en
http://csirc.ugr.es/informatica/correoelectronico/SolicitarCuentaCorreo.html

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.

Leer las actividades que se van a realizar durante las horas de prácticas en las aulas de ordenadores (ver página 12 de este guión).

Intentad resolver previamente los ejercicios enumerados a continuación. Escribid las soluciones en los ficheros `.cpp` indicados y guardarlos en un *pendrive* o en la nube.

1. (`voltaje.cpp`)

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

   voltaje = intensidad * resistencia

2. (`circunferencia.cpp`)

   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:

   $$\text{longitud circunferencia} = 2\pi r \qquad \text{área circulo} = \pi r^2$$

   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 `3.1415927`, recompilad y ejecutad.

3. (`salario.cpp`)

   Indique cuál sería el resultado de las siguientes operaciones:

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

   salario_base = 1000;
   salario_final = salario_base;
   incremento = 200;

   salario_final = salario_final + incremento;
   salario_base = 3500;

   cout << endl;
   cout << "Salario base: " << salario_base << endl;
   cout << "Salario final: " << salario_final << endl;
   ```

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

► Actividades a realizar en las aulas de ordenadores

Durante esta sesión de prácticas, el profesor guiará a los alumnos en el desarrollo de las siguientes actividades que se desarrollarán en el aula de prácticas:

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 no podrá funcionar.

El alumno deberá crear el directorio (`U:\FP`).

Si durante la sesión se requiere abrir algún fichero, éste puede encontrarse en **PRADO**.

Muy Importante. No emplearemos los ficheros que se encuentran en la unidad `H:`. Si usted decidiera hacerlo, sepa que esos ficheros están protegidos y no pueden modificarse. Por tanto, habrá que copiarlos a la unidad local `U:`, dónde sí podrán ser modificados.

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

El primer programa

Copiando el código fuente

En la sección `PRÁCTICA 1` de **PRADO** encontrará en la carpeta `Archivos Sesión 1` el fichero `pitagoras.cpp`.

Cread una carpeta en `U:\FP` llamada `SESION_01` y copiar en ella el fichero `pitagoras.cpp`.

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

```
U:\FP\SESION_01
```

y haced doble click sobre el fichero `pitagoras.cpp`. Debe aparecer una ventana como la de la figura 1:

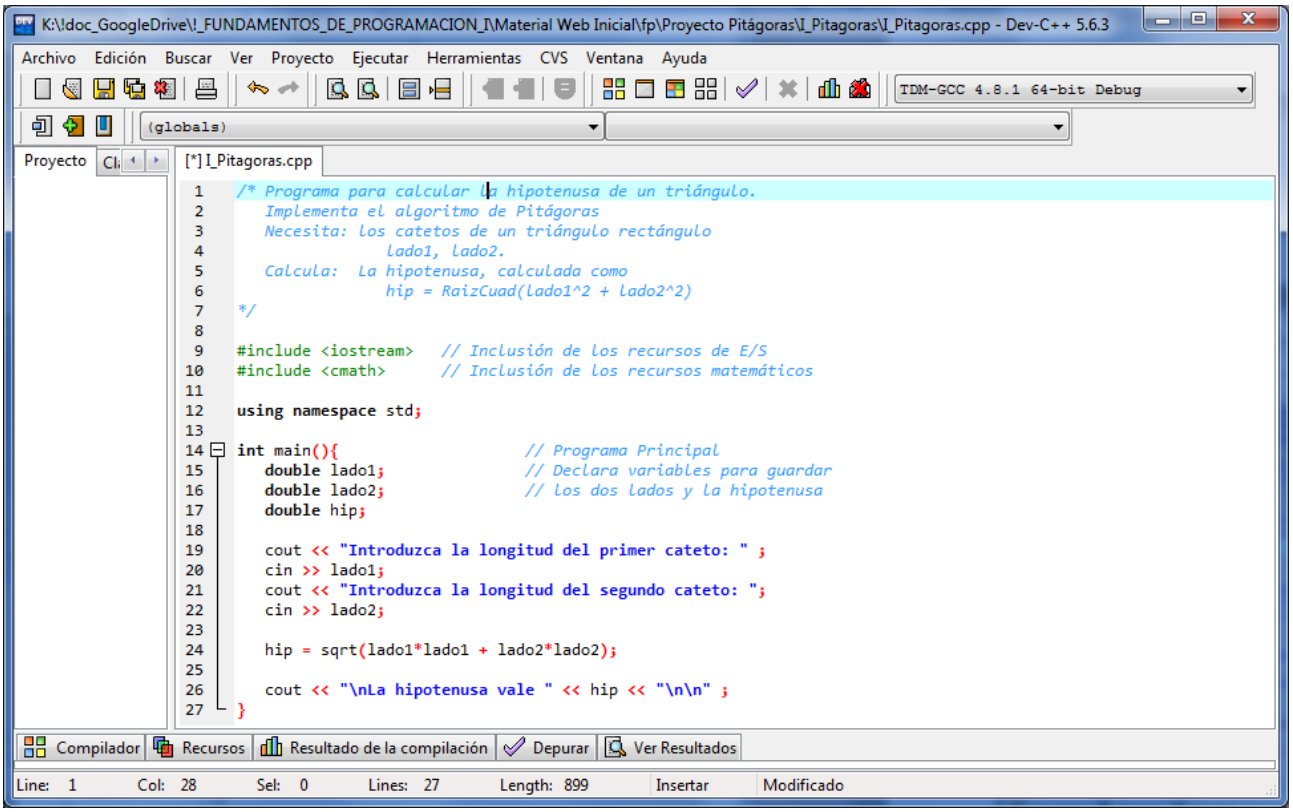


Figura 1: Programa que implementa el algoritmo de Pitágoras

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 órdenes que usan `cout` y `cin` respectivamente.

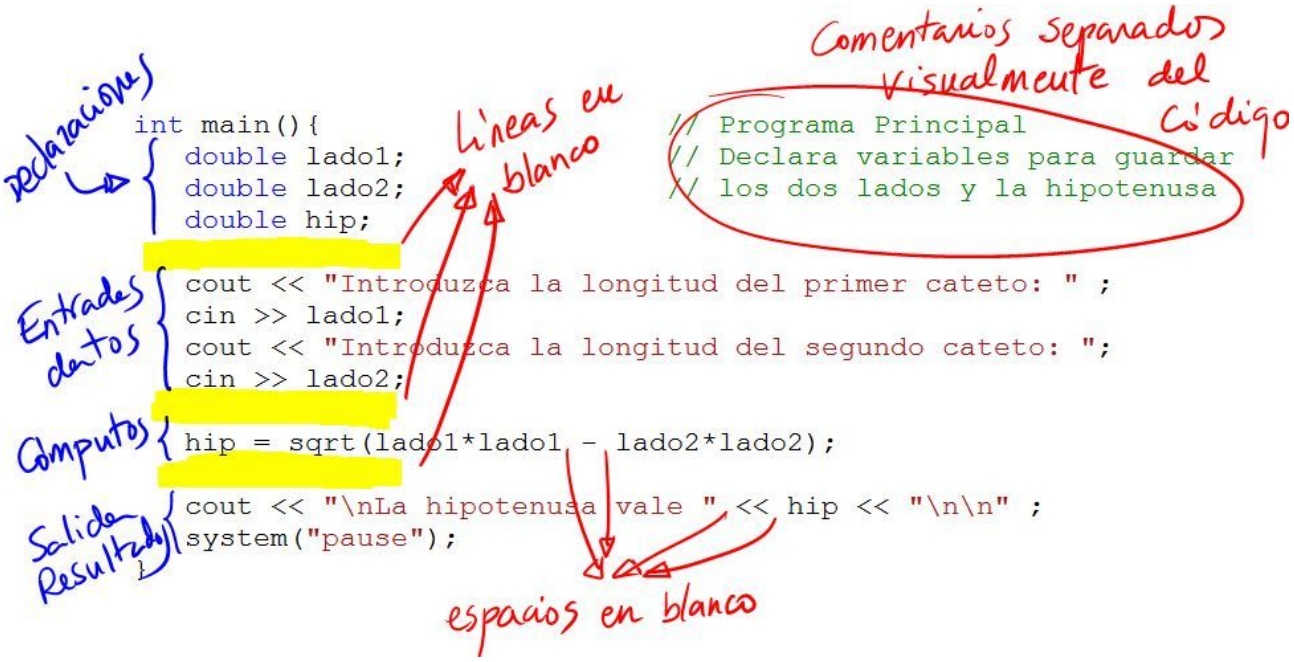


Figura 2: Escritura de código

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

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, tendremos el fichero `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 consola en la que se

estará ejecutando el programa. La ejecución del programa se detendrá en aquellos puntos del mismo donde se requiera la intervención del usuario, es decir, en la operaciones de entrada de datos a través del dispositivo estándar de entrada. En este ejemplo, sería en las dos operaciones `cin`. En el resto de los casos, la ejecución del programa continuará hasta el final. La introducción de datos mediante la sentencia `cin` se hace siempre de la misma manera; primero se introduce el valor que se desee y al terminar se pulsa la tecla RETURN.

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

Como hemos indicado anteriormente, en la fase de generación del ejecutable se ha creado un fichero en el Sistema que se llama igual que nuestro fichero pero sustituyendo la extensión ”cpp” por ”exe”, es decir, `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 `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 `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 `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. Borrad 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. Borrad alguna de las dobles comillas en una constante de cadena de caracteres, tanto las comillas iniciales como las finales.
9. Borrad alguna de las llaves que delimitan el inicio y final del programa.
10. Borrad 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 `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. El primer ejemplo que vamos a implementar corresponde al ejercicio 1 sobre la Ley de Ohm, propuesto en la *Actividad: Resolución de Problemas* de esta sesión de prácticas.

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

`Archivo->Nuevo Código Fuente (Ctr-N)`

Para cambiar el nombre asignado por defecto, seleccionamos `Archivo -> Guardar Como`. Seleccionamos la carpeta `U:\FP\SESION_01` e introducimos el nombre del nuevo fichero fuente: `voltaje` (no es necesario escribir la extensión `.cpp`).

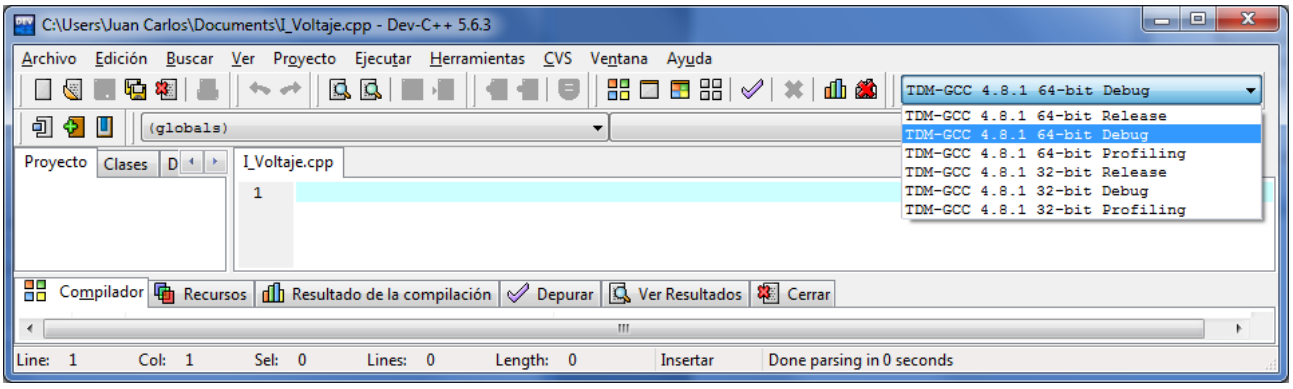


Figura 3: Creación de un programa nuevo

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

`TDM-GCC ... Debug`

Ya estamos en condiciones de resolver el problema pedido. Escribimos el código en la ventana de edición.
Este ejercicio sigue un esquema similar al del ejercicio que calculaba la longitud de la hipotenusa de un triángulo rectángulo cuando se conocen las longitudes de los dos catetos:

1. Entrada: leer (desde teclado) los valores de entrada (intensidad y resistencia),
2. Cálculos: Aplicar la fórmula que relaciona los datos de entrada con la salida (voltaje).
3. Salida: Mostrar el resultado.

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.

Sesión 2

Problemas básicos (entrada / cálculos / salida) (I)

► Actividades a realizar en casa

Actividad: Lectura de soluciones de ejercicios resueltos.

Lea la solución de los siguientes ejercicios, que podrá encontrar en PRADO (sección dedicada a la Sesión 2 de Prácticas.

2 (Ley de Ohm) `I_Voltaje.cpp`
4 (Aproximaciones de π (1)) `I_Aprox_PI.cpp`
5 (Aproximación de π (2)) `I_Aprox_PI_ArcoSeno.cpp`
6 (Conversión de grados a radianes) `I_Grados_a_Radianes_ctes.cpp`
10 (Interés bancario) `I_Interes.cpp`
13 (Circunferencia -con constantes-) `I_Circunferencia_ctes.cpp`

Actividad: Resolución de problemas.

Resolved los ejercicios siguientes de la Relación de Problemas I (página RP-I.1).

- *Obligatorios:*

  14 (Conversión millas-km y viceversa) `I_Millas_Km.cpp`
  15 (Conversión de unidades de distancia) `I_ConversionUnidadesDistancia.cpp`
  17 (Intercambiar dos variables) `I_Intercambio.cpp`
  18 (Ajustar horas, minutos y segundos) `I_AjusteTiempo.cpp`
  19 (Tiempo entre dos instantes) `I_DiferenciaTiempo.cpp`
  20 (Ritmo y velocidad) `I_RitmoVelocidad.cpp`
  22 (Saltos en intervalos) `I_Desplazamiento_Intervalo.cpp`
  24 (Gaussiana) `I_Gaussiana.cpp`

  Recordad que antes del inicio de esta sesión en el aula de ordenadores hay que entregar las soluciones en **PRADO** siguiendo las instrucciones y en el plazo indicado en la convocatoria de la entrega.

- *Opcionales:*
  7 (Tarifas aéreas) `I_Tarifa_Aerea_Km.cpp`
  8 (Tarifas aéreas con descuento) `I_TarifaAerea_descuento_ctes.cpp`
  9 (Variación porcentual) `I_VariacionPorcentual.cpp`
  11 (Incremento porcentual de salario) `I_IncrementoSalario.cpp`
  12 (Reparto de beneficios) `I_RepartoBeneficios.cpp`
  16 (Consumo de gasolina) `I_ConsumoGasolina.cpp`
  23 (Media aritmética) `I_Media.cpp`

► *Actividades a realizar en las aulas de ordenadores*

Durante esta sesión de prácticas se discutirá sobre las soluciones a las actividades propuestas, y se trabajará sobre actividades adicionales que proponga el profesor.

► *Recomendaciones*

Todas las soluciones deben estar correctamente escritas y **comentadas**.

► *Normas detalladas*

Deberá entregar únicamente el fichero `FP_sesion02.zip` (**importante:** no `.rar` ni otro formato, sólo `.zip`), resultado de empaquetar todos los ficheros fuente (`.cpp`), uno por cada ejercicio resuelto.

Los nombres de los ficheros fuente deben ser los indicados en la lista de ejercicios propuestos (página 21).

La entrega se realizará en la plataforma web de la asignatura, en el periodo habilitado para ello (consulten su correo electrónico o la sección **novedades**).

Sesión 3

Problemas básicos (entrada / cálculos / salida) (II)

► Actividades a realizar en casa

Actividad: Lectura de soluciones de ejercicios resueltos.

Lea la solución de los siguientes ejercicios, que podrá encontrar en PRADO (sección dedicada a la Sesión 3 de Prácticas.

21 (Lectura simple de intervalo) `I_Intervalo.cpp`
23 (Media aritmética) `I_Media.cpp`
26 (Detectar y separar dígitos) `I_PintaDigitos.cpp`
37 (Distancia Euclídea) `I_DistanciaEuclidea.cpp`

Actividad: Resolución de problemas.

Resolved los ejercicios siguientes de la Relación de Problemas I (página RP-I.1).

Recordad que antes del inicio de esta sesión en el aula de ordenadores hay que entregar las soluciones en **PRADO** siguiendo las instrucciones y en el plazo indicado en la convocatoria de la entrega.

- *Obligatorios:*

  9 (Cifrado de caracteres) `I_CifradoRotacion.cpp`
  25 (Pasar de mayúscula a minúscula) `I_Mayuscula_a_Minuscula.cpp`
  32 (Expresiones lógicas) `I_Expresiones_Logicas.cpp`
  35 (Mínimo número de bits) `I_Min_Num_Bits.cpp`
  36 (Truncar decimales) `I_TruncaDecimales.cpp`

- *Opcionales:*

  33 (PVP) `I_PVP_Automovil.cpp`
  38 (Distancia entre dos puntos) `I_Haversine.cpp`
  39 (Expresiones) `I_Expresiones.cpp`
  40 (Área de un triángulo) `I_AreaTriangulo.cpp`

► Recomendaciones

Todas las soluciones deben estar correctamente escritas, **comentadas** y **modularizadas**.

► Normas detalladas

Deberá entregar únicamente el fichero `FP_sesion03.zip` (**importante:** no `.rar` ni otro formato, sólo `.zip`), resultado de empaquetar todos los ficheros fuente (`.cpp`), uno por cada ejercicio resuelto.

Los nombres de los ficheros fuente deben ser los indicados en la lista de ejercicios propuestos (página 23).

La entrega se realizará en la plataforma web de la asignatura, en el periodo habilitado para ello (consulten su correo electrónico o la sección **novedades**).

► Actividades a realizar en las aulas de ordenadores

Mientras el profesor corrige individualmente los ejercicios de algunos alumnos, realizar la actividad siguiente y trabajar sobre los ejercicios opcionales.

Durante el resto de la sesión se discutirá sobre las soluciones propuestas a los ejercicios de esta sesión.

Redireccionando las entradas de `cin` y las salidas a `cout`

Cada vez que se ejecuta una instrucción `cin >>` , se lee un dato desde el periférico de entrada por defecto (el teclado): el usuario escribe el valor del dato y confirma/termina pulsando la tecla ENTER.

Pueden introducirse los valores de varios datos (incluso de diferentes tipos) en una única instrucción de lectura. Tan sólo hay que escribir los valores de los datos separados por espacios en blaco y tabuladores (caracteres *separadores*). Para comprobarlo, copiad el fichero `Redireccion_cin.cpp` desde `PRADO`. Observad el código del programa y ejecutadlo. Para introducir los valores de los datos pedidos (un entero y dos caracteres) empezaremos por escribir los dígitos del entero, un número arbitrario de espacios en blanco y/o tabuladores, un carácter, un número arbitrario de espacios en blanco y/o tabuladores, otro carácter y finalmente, ENTER.

Para comprobar el correcto funcionamiento de nuestros programas, habitualmente 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 `copiar-pegar`. Para comprobarlo, cread un fichero *de texto* con un entero y dos caracteres. Separad estos tres datos con varios espacios en blanco, por ejemplo. 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 desde 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 aparece en la instrucción de salida `cout <<` de C++).

Para probarlo, descargad desde `decsai` el fichero

`Redireccion_cin_datos_entrada.txt`

y copiadlo en la misma carpeta en la que se ha descargado el programa `Redireccion_cin.cpp`. Abrimos dicha carpeta desde el explorador y seleccionamos con el click derecha del ratón `Abrir Símbolo del Sistema`[1]

Una vez nos hemos situado en el directorio que contiene el ejecutable y el fichero de datos, introducimos la instrucción siguiente:

```
Redireccion_cin.exe < Redireccion_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 y no se empleará el teclado para la introducción de datos.

Hay que destacar que este redireccionamiento de la entrada lo estamos haciendo en la llamada al ejecutable desde la consola del sistema operativo[2].

También es posible redireccionar la salida para que cada vez que se ejecute una instrucción `cout <<` en el programa, el texto se "envíe" a un fichero de texto en lugar de a la consola. En este caso cuando escribimos la instrucción siguiente:

```
programa.exe > resultado.txt
```

todo lo que se envía a `cout` en `programa` es enviado al fichero `resultado.txt` en lugar de ser mostrado en la consola (el programa se ejecuta *en silencio*).

Finalmente, es posible redireccionar la entrada y salida simultaneamente.

```
programa.exe < entrada.txt  > salida.txt
```

[1] 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`

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

Sesión 4

Estructuras condicionales

► Actividades a realizar en casa

Actividad: Lectura de soluciones de ejercicios resueltos.

Lea la solución de los siguientes ejercicios, que podrá encontrar en PRADO (sección dedicada a la Sesión 4 de Prácticas.

- El programa `II_Gaussiana_Comp.cpp` es una ampliación del ejercicio 24 (Gaussiana, con comprobación) de la Relación de Problemas I (página RP-I.1). Incluye las instrucciones condicionales adecuadas en la **entrada de datos** para evitar realizar cálculos cuando los valores suministrados por el usuario sean incorrectos (en este caso, el valor de la desviación típica).
- Los programas siguientes son de la Relación de Problemas II (página RP-II.1):

  1 (Subir sueldo) `II_Subir_Sueldo.cpp`
  2 (Ver si dos números se dividen) `II_SeDividen.cpp`
  7 (Comprobar si un año es bisiesto) `II_Bisiesto.cpp`
  10 (Velocidad imputada) `II_Velocidad_Imputada.cpp`
  12 (Pertenencia a intervalo) `II_Pertenece_Intervalo.cpp`

- Estudie el seminario sobre los **tipos de datos definidos por enumeración** (también explicados en los apuntes, sección 1.7.6 del Tema II) y lea la solución del siguiente ejercicio (disponible en PRADO, sección dedicada a la Sesión 4 de Prácticas):

  19 (Tres números ordenados - enumerado) `II_EstanOrdenados_enum.cpp`

Actividad: Resolución de problemas.

Ampliar los ejercicios de la Relación de Problemas I (página RP-I.1) que se enumeran a continuación. Debe incluir las instrucciones condicionales adecuadas en la **entrada de datos** para evitar realizar cálculos cuando los valores suministrados por el usuario sean incorrectos, y en ese caso debe mostrar un mensaje de **error** adecuado antes de finalizar la ejecución del programa.

Todos los ejercicios indicados a continuación son **obligatorios**.

25 (Pasar de mayúscula a minúscula, con comprobación)
`II_Mayuscula_a_Minuscula_Comp.cpp`
26 (Detectar y separar dígitos, con comprobación)
`II_PintaDigitos_Comp.cpp`
35 (Mínimo número de bits, con comprobación)
`II_Min_Num_Bits_Comp.cpp`
36 (Truncar decimales, con comprobación) `II_TruncaDecimales_Comp.cpp`

Resolved los ejercicios siguientes de la Relación de Problemas II (página RP-II.1):

- *Obligatorios:*

  3 (Comprobar ordenación) `II_EstanOrdenados.cpp`
  5 (Tarifa aérea con descuentos) `II_TarifaAerea_Km_Descuentos.cpp`
  6 (Tiempo entre dos instantes -II-) `II_DiferenciaTiempo.cpp`
  9 (Cifrado de caracteres -II-) `II_CifradoRotacion.cpp`
  16 (Tarifa aparcamiento) `II_Parking.cpp`
  18 (Conversión de unidades de medida) `II_ConversionUnidadesMedida.cpp`

  Resuelva, además, los siguientes ejercicios usando algún tipo **enumerado**:

  20 (Subir salario precario - enumerado) `II_Subida_SalarioPrecario_enum.cpp`
  21 (Conversión entre caracteres - enumerado) `II_May_a_Min_viceversa_enum.cpp`

- *Opcionales:*

  8 (Haversine, cáculo del mínimo) `II_Haversine_con_minimo.cpp`
  11 (Calcula sanción tráfico) `II_Consulta_Multa.cpp`
  14 (Descuento por ventas) `II_Descuento_Ventas.cpp`
  15 (Reajustar renta) `II_ReajustarRenta.cpp`
  17 (Tarifa aparcamiento (Ampliación)) `II_Parking_Ampliacion.cpp`

► *Recomendaciones*

Todas las soluciones deben estar correctamente escritas, **comentadas** y **modularizadas**.

► Normas detalladas

Deberá entregar únicamente el fichero `FP_sesion04.zip` (**importante:** no `.rar` ni otro formato, sólo `.zip`), resultado de empaquetar todos los ficheros fuente (`.cpp`), uno por cada ejercicio resuelto.

Los nombres de los ficheros fuente deben ser los indicados en la lista de ejercicios propuestos (página 28).

La entrega se realizará en la plataforma web de la asignatura, en el periodo habilitado para ello (consulten su correo electrónico o la sección **novedades**).

► Actividades a realizar en las aulas de ordenadores

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 esta sesión de prácticas emplearemos como base el programa `II_Media_int_ampliado.cpp`, disponible en `PRADO`.

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

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:

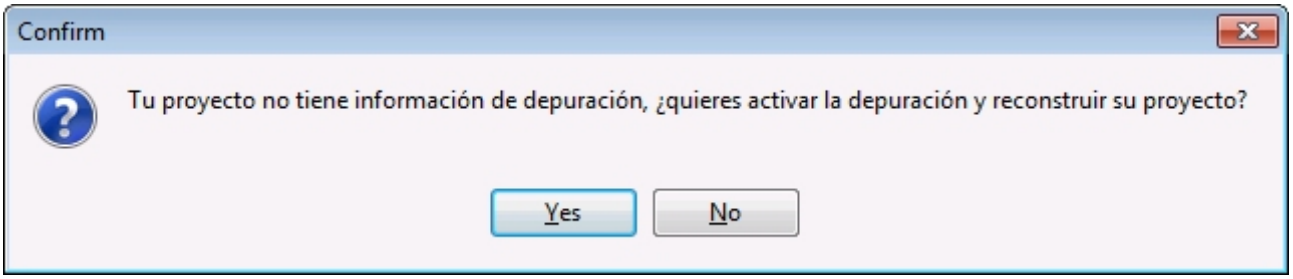


Figura 4: Ventana emergente que aparece cuando la configuración actual del compilador no permite tareas de depuración

1. Establecer una línea del programa en la que queremos que se pare la ejecución. Lo haremos introduciendo un **punto de ruptura** o (*breakpoint*) en dicha línea. Si sospechamos dónde puede estar el error, situaremos el punto de ruptura en dicha línea. En caso contrario, lo situaremos:

   a) al principio del programa, si no sabemos exactamente dónde falla el programa, o

   b) al principio del bloque de instrucciones del que desconfiamos, siempre y cuando tengamos confianza en todas las instrucciones que se ejecutan antes.

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

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

2. Comenzar la depuración:

   a) pulsar **F5**,

   b) pulsar sobre el icono  ,

   c) seleccionar en el menú `Ejecutar | Depurar`, ó

   d) en la zona inferior, pestaña `Depurar`, pulsar el botón **Depurar** (figura 6)

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

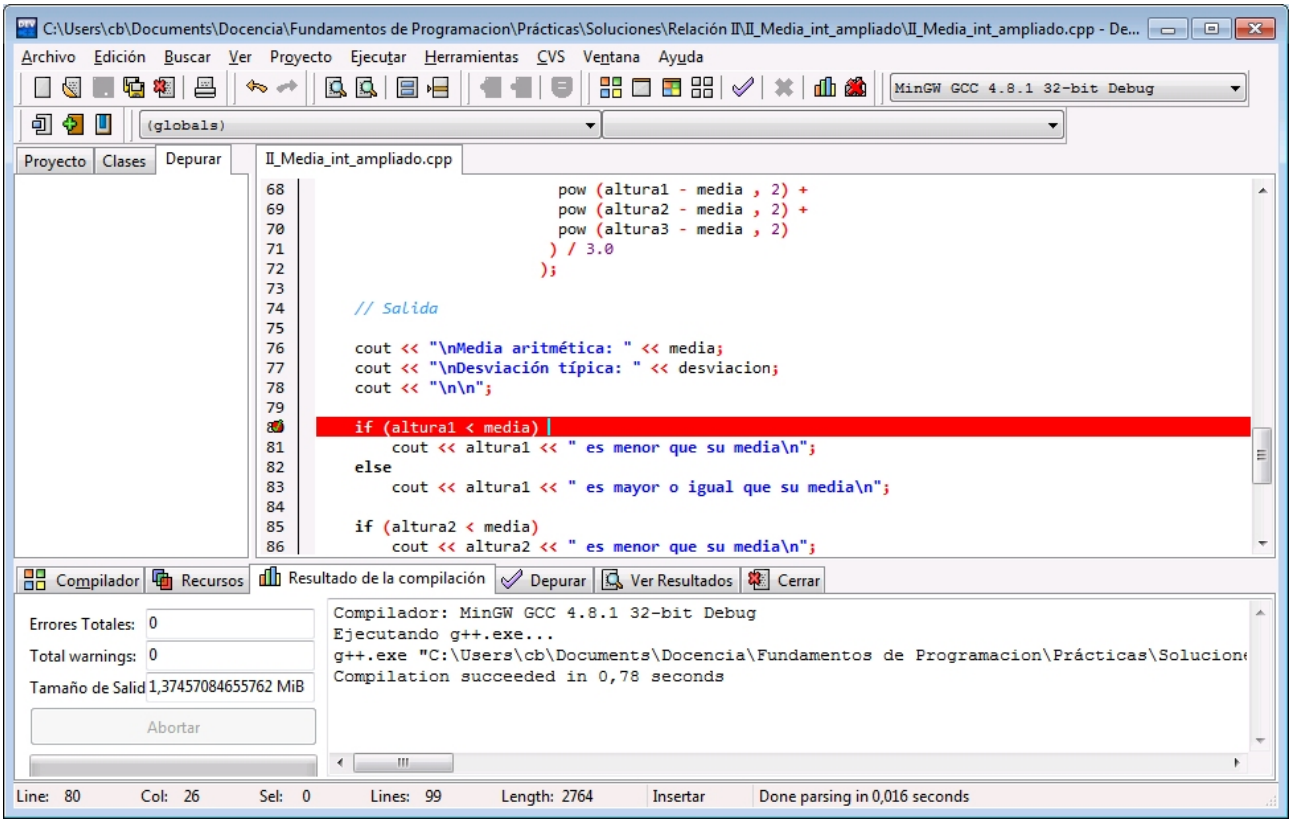


Figura 5: Se ha activado un punto de ruptura

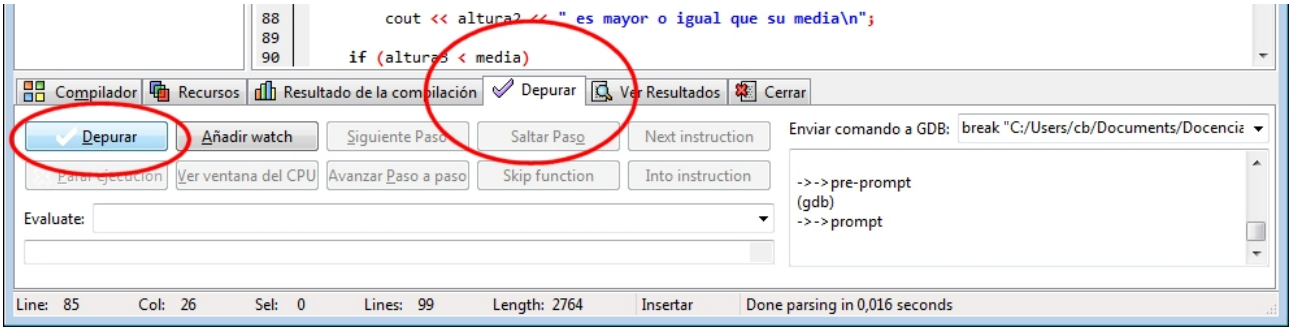


Figura 6: Inicio del proceso de depuración

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).

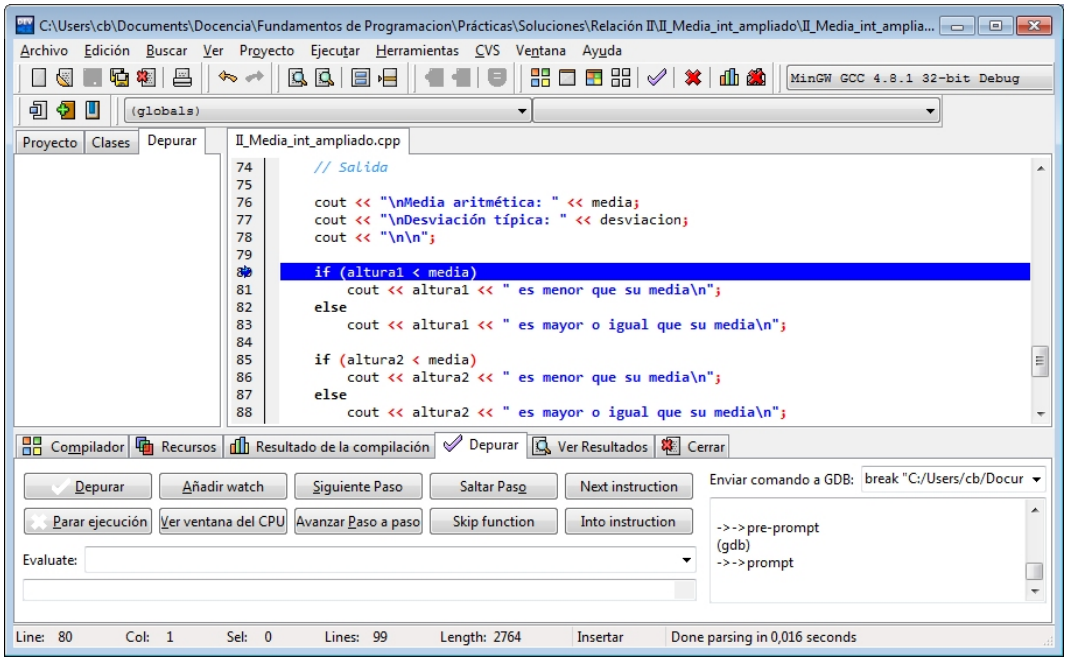


Figura 7: Inicio del proceso de depuración

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**: Ejecuta la siguiente instrucción. Si se trata de una llamada a una función, la ejecuta y continúa con la siguiente instrucción, sin entrar a ejecutar las instrucciones internas de la función.
- **Avanzar Paso a paso**: 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.

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

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

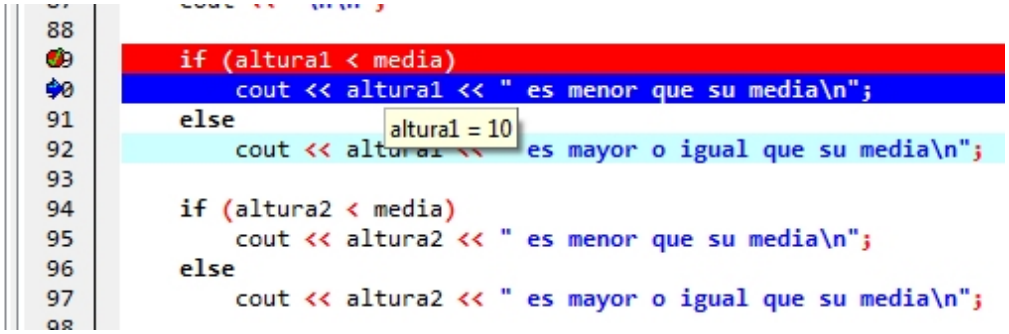


Figura 8: Inspeccionando el valor de una variable

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

Para añadir una variable podemos:

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

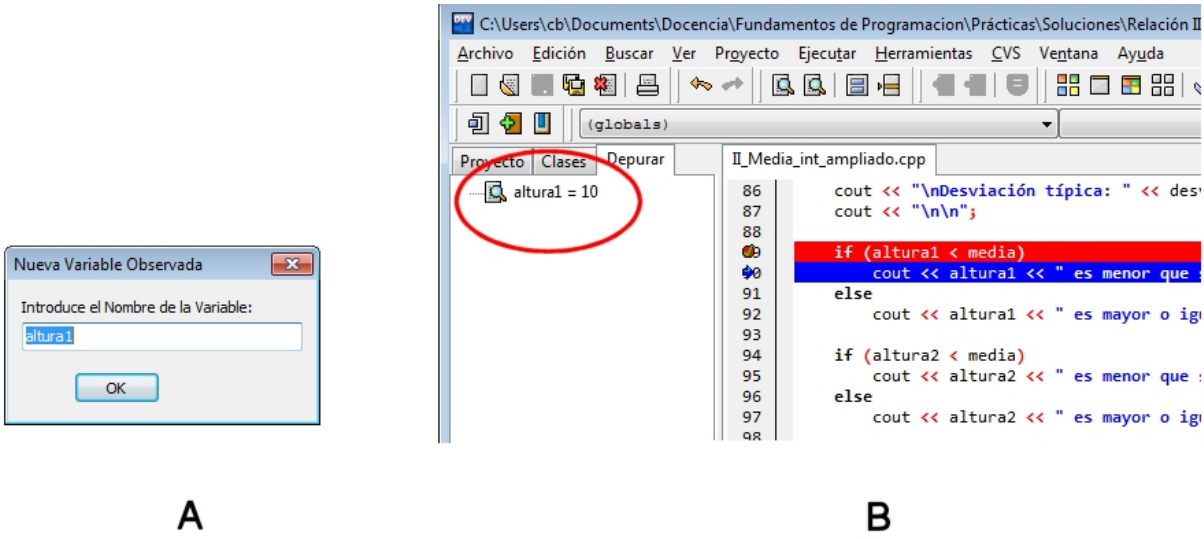


Figura 9: Añadiendo una variable para su inspección permanente

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

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

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

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

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

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

Sesión 5

Estructuras repetitivas (I)

► Actividades a realizar en casa

Actividad: Lectura de soluciones de ejercicios resueltos.

Lea la solución de los siguientes ejercicios sobre ciclos, que podrá encontrar en PRADO (sección dedicada a la Sesión 5 de Prácticas:

22 (Conversión a minúsculas - filtros) `II_Mayuscula_Minuscula_FiltroEntrada.cpp`
23 (Tiempo entre dos instantes -III-)`II_DiferenciaTiempo_FiltroEntrada.cpp`
25 (Tarifa aérea -III-) `II_TarifaAerea_Km_Descuentos_FiltroEntrada.cpp`
29 (Reinvertir inversión) `II_Interes_Reinvierte.cpp`
35 (Calcular años bisiestos entre dos dados) `II_Bisiestos_Intervalo.cpp`
39 (Sucursal con más ventas) `II_Sucursales.cpp`

Actividad: Resolución de problemas.

Deberá resolver los siguientes ejercicios sobre **ciclos** de la Relación de Problemas II (página RP-II.1):

- *Obligatorios:*

  28 (Pinta dígitos general separando E/C/S) `II_PintaDigitos_General_ECS.cpp`
  30 (Doblar inversión) `II_Interes_Doblar.cpp`
  31 (Parking: calcular tiempo de estancia) `II_Parking_LimitadoDinero.cpp`
  43 (Mayor secuencia ordenada -datos en un rango establecido-)
  `II_SecOrdenadaMasLarga_Rango.cpp`
  44 (Control carrera) `II_Carrera.cpp`
  49 (Acertar un número) `II_Adivinar.cpp`
  51 (Calcular CDF de la gaussiana) `II_CDF_Gaussiana.cpp`
  52 (Aproximación a π -Madhava-) `II_Aprox_PI_Madhava.cpp`

- *Opcionales:*

  26 (Divisores de un número) `II_Divisores.cpp`
  45 (Método RLE) `II_RLE.cpp`
  53 Aproximación a π -Gregory y Leibniz-) `II_Aprox_PI_Gregory-Leibniz.cpp`
  54 Aproximación a π -Wallis-) `II_Aprox_PI_Wallis.cpp`

► *Recomendaciones*

Todas las soluciones deben estar correctamente escritas, **comentadas** y **modularizadas**.

► *Normas detalladas*

Deberá entregar únicamente el fichero `FP_sesion05.zip` (**importante:** no `.rar` ni otro formato, sólo `.zip`), resultado de empaquetar todos los ficheros fuente (`.cpp`), uno por cada ejercicio resuelto.

Los nombres de los ficheros fuente deben ser los indicados en la lista de ejercicios propuestos (página 35).

La entrega se realizará en la plataforma web de la asignatura, en el periodo habilitado para ello (consulten su correo electrónico o la sección **novedades**).

► *Actividades a realizar en las aulas de ordenadores*

En esta sesión se discutirá sobre las soluciones propuestas a los ejercicios de esta sesión.

UNIVERSIDAD DE GRANADA

Departamento de Ciencias de la Computación e Inteligencia Artificial

Fundamentos de Programación

Doble Grado en Ingenieria Informática y Matemáticas

Francisco José Cortijo Bon

cb@decsai.ugr.es

Relaciones de Problemas

Curso 2020 - 2021

Departamento de Ciencias de la Computación e Inteligencia Artificial

Universidad de Granada

RELACIÓN DE PROBLEMAS I. Introducción a C++

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

```
int salario_bas
int salario_final;
int incremento;

salario_base = 1000;
salario_final = salario_base;

incremento = 200;
salario_final = salario_final + incremento;

salario_base = 3500;

cout << "\nSalario base: " << salario_base;

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

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

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

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

   voltaje = intensidad * resistencia

   Finalidad: Asignar a una variable el resultado de una expresión. Dificultad Baja.

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

   $$\text{longitud circunferencia} = 2\pi r \qquad \text{área circulo} = \pi r^2$$

   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 `3.1415927`, recompilad y ejecutad.

   Finalidad: Conciernarnos del problema de los literales repetidos a lo largo del código. Dificultad Baja.

4. Escriba un programa que muestre en pantalla el resultado de las siguientes expresiones numéricas que constituyen una aproximación al valor de π. La primera es del año 1800 antes de Cristo, la segunda (también de la misma era) es una aproximación introducida por los matemáticos mesopotámicos y la tercera es del siglo II (introducida por Claudio Ptolomeo)

$$\pi \approx \frac{256}{81} \qquad \pi \approx 3 + \frac{1}{8} \qquad \pi \approx \frac{377}{120}$$

   El resultado debe ser `3.16049`, `3.125` y `3.14167`

   Finalidad: Mezclar tipos enteros y reales. Dificultad Baja.

5. El número π es un número irracional y por tanto tiene infinitas cifras decimales.

   Hay diversos métodos para obtener este valor, algunos mejores que otros. Una forma simple de obtenerlo es a través de la función arco-seno, ya que $\pi/6$ es el ángulo (en radianes) cuyo seno vale de 0.5. Matemáticamente se expresa como

$$\frac{\pi}{6} = \text{arco-seno}(0{,}5)$$

   Construir un programa que imprima el valor de π calculado a partir de la anterior fórmula.

   El cálculo del arco-seno se realiza en C++ con la función `asin` de la biblioteca `cmath`.

   Dificultad Baja.

6. Queremos transformar una cantidad g dada en grados a radianes. Para ello, basta recordar que π radianes equivale a 180^o por lo que

$$r = g\,\frac{\pi}{180}$$

   Construya un programa que lea un número real (grados) y muestre los radianes correspondientes.

   Para el valor de π puede usar un literal o bien las expresiones de los ejercicios 4 ó 5.

   Dificultad Baja.

7. 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. A continuación se suman 10 céntimos por cada kilómetro de distancia al destino.

   Cree un programa que lea el número de kilómetros al destino y calcule el precio final del billete.

   Dificultad Baja.

8. La compañía aérea del ejercicio 7 quiere aplicar una política de descuentos al precio del billete. Amplíe el programa anterior para después de imprimir el precio del billete pida un porcentaje de descuento (dato `double`) y muestre el precio final después de aplicar el descuento indicado.

   Dificultad Baja.

9. Se define el concepto variación porcentual como:

$$VP = abs\left(100 \times \frac{\text{valor final} - \text{valor inicial}}{\text{valor inicial}}\right)$$

   donde abs calcula el valor absoluto (`cmath`).

   Escriba un programa en C++ que lea el valor inicial y final de un producto (variables de tipo `double`) y calcule la variación porcentual del mismo.

   Dificultad Baja.

10. 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:

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

    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 interés 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?

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

11. Construya un programa para leer el valor de una variable `salario_base` de tipo `double`, la incremente un 2 % e imprima el resultado en pantalla. Para realizar este cálculo, multiplique por `1.02` el valor original. Tiene varias alternativas:

    a) Calcular `1.02 * salario_base` dentro de la sentencia `cout`

b) Introducir una variable `salario_final`, asignarle la expresión anterior y mostrar su contenido en la sentencia `cout`

c) Modificar la variable original `salario_base` con el resultado de incrementarla un 2 %.

Indique qué alternativa elige y justifíquela.

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

12. 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 cuanto 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 guardar la ganancia total y

    a) variables `double`

    b) variables `int`

    para guardar las ganancias del diseñador y fabricantes,

    Muestre los resultados y analice los resultados obtenidos.

    Importante: No debe repetir cálculos ya realizados.

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

13. Modificar el programa que resuelve el ejercicio 3 sustituyendo los literales que hacen referencia al valor de π por una *constante* llamada `PI`.

    Modifique su valor, recompile y ejecute. ¿Es esta solución mejor que la del ejercicio 3? ¿por qué?.

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

14. Escriba un programa que lea un distancia en millas (como un real) y muestre la cantidad equivalente en kilómetros. A continuación leerá una distancia en kilómetros (como un real) y mostrará la cantidad equivalente en millas.

    Debe tener en cuenta que 1 milla equivale a `1.609` kilómetros.

    Finalidad: Asignar a una variable el resultado de una expresión. Dificultad Baja.

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

1 pulgada= 25,4 milímetros	1 yarda = 0,9144 metros
1 pie = 30,48 centímetros	1 milla = 1,609344 kilómetros

Finalidad: Plantear la solución con una doble conversión. *Dificultad Baja.*

16. 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.

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

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

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

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

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

    Estaríamos tentados a escribir el siguiente código:

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

    pero no funciona correctamente ¿Por qué?

    Proponga una solución e impleméntela.

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

18. Leed desde teclado tres variables correspondientes a un número de horas, minutos y segundos, respectivamente. Diseñar un algoritmo que calcule las horas, minutos y segundos dentro de su rango correspondiente. Por ejemplo, dadas 10 horas, 119

minutos y 280 segundos, debería dar como resultado 12 horas, 3 minutos y 40 segundos. En el caso de que nos salgan más de 24 horas, daremos también los días correspondientes (pero ya no pasamos a ver los meses, años, etc)

Como consejo, utilizad los operadores `/` (representa la división entera cuando los dos argumentos son enteros) y `%` (representa el resto de la división entera).

Finalidad: Usar expresiones y variables para no repetir cómputos. Dificultad Media.

19. 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 Media.

20. 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 variables de diferentes tipos. Dificultad Baja.

21. Un intervalo es un espacio métrico comprendido entre dos valores o cotas, a y b, siendo a la cota inferior y b la cota superior. Cada extremo de un intervalo puede ser abierto o cerrado. En este problema no se consideran intervalos con extremos infinitos.

    Construya un programa que lea e imprima los datos de un intervalo. Para ello, el programa debe leer los datos en el siguiente orden:

    a) Un carácter que represente el tipo de intervalo por la izquierda: el usuario deberá introducir el carácter `(` si es abierto y `[` si es cerrado.

    b) Un número real con la cota izquierda.

    c) El carácter `,` como separador de las dos cotas

    d) Un número real con la cota derecha.

    e) Un carácter que represente el tipo de intervalo por la derecha: el usuario deberá introducir el carácter `)` si es abierto y `]` si es cerrado.

El programa mostrará en pantalla el mismo intervalo introducido.

Finalidad: Manejar entrada de datos de distinto tipo. Dificultad Baja.

22. Queremos construir una expresión numérica que desplace un entero un número dado de posiciones, pero lo mantenga dentro de un intervalo.

    Por ejemplo, si el intervalo fijado es $[65, 90]$, el desplazamiento es de 3 unidades y el entero a desplazar es el 70, el resultado sería 73. Si el entero fuese el 89 y el desplazamiento 3, el resultado sería 92. Al no estar el 92 dentro del intervalo, se realiza un *ciclo* de forma que el 91 se transformaría en el 65 y el 92 en el 66.

    Se pide construir un programa que lea dos enteros `minimo` y `maximo` que determinarán el intervalo [`minimo`, `maximo`] (supondremos que el usuario introduce como `maximo` un valor mayor o igual que `minimo`). A continuación el programa leerá un valor entero `desplazamiento` (supondremos que el usuario introduce un valor entre 0 y `maximo` - `minimo`). Finalmente, el programa leerá un entero `a_desplazar` (supondremos que el usuario introduce un número entre `minimo` y `maximo`). El programa sumará al valor leido el `desplazamiento` y lo convertirá en un entero dentro del intervalo [`minimo`, `maximo`] tal y como se ha descrito anteriormente.

    Finalidad: Trabajar con los operadores enteros. Dificultad Media.

23. Redactar un algoritmo para calcular la media aritmética muestral y la desviación estándar (o típica) muestral de las alturas de **tres** personas. Éstos valores serán reales (de tipo `double`). La fórmula general para un valor arbitrario de `n` es:

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

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

    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*, en inglés) nos da una idea de la dispersión de éstos.

    Ejecutad el programa con varios valores y comprobad que el resultado es correcto utilizando una calculadora científica o cualquier calculadora online como por ejemplo la disponible en http://www.disfrutalasmatematicas.com/datos/desviacion-estandar-calculadora.html

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

24. 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, respectivamente). Su definición viene dada por la expresión:

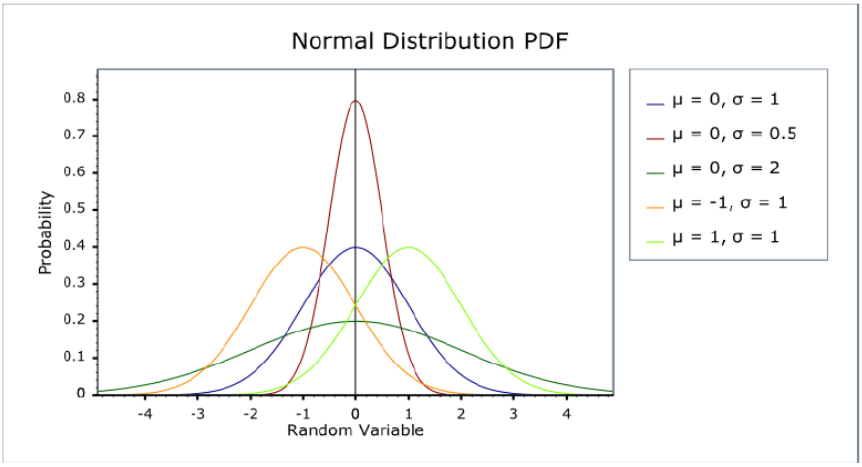
$$\text{gaussiana}(x) = \frac{1}{\sigma\sqrt{2\pi}} \; e^{\left\{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right\}}$$

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

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

- Para elevar el número e a un valor cualquiera, use la función `exp`. Por ejemplo, para calcular e^8 debería usar la expresión `exp(8)`
- Para calcular la raíz cuadrada, use `sqrt`. Por ejemplo, para calcular $\sqrt{8}$ debería usar la expresión `sqrt(8)`
- Para elevar un número a otro, utilice la función `pow` en la siguiente forma: `pow(base, exponente)`. Por ejemplo, para calcular 2^{10} debería usar la expresión `pow(2,10)`

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



Comprobad que los resultados son correctos, usando:
https://www.easycalculation.com/statistics/normal-pdf.php

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

25. 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 entre tipos enteros y de carácter. Dificultad Baja.

26. Escribir un programa que lea un valor entero en un dato de tipo `string`. A continuación lo convierte y asigna a un dato `int` (supondremos que el usuario introduce **siempre** un entero de **tres dígitos**, como por ejemplo 351).

    Escribid en pantalla los dígitos separados por dos espacios en blanco. Con el valor anterior la salida sería:

    ```
    3  5  1
    ```

    Lo que se muestra en pantalla es el contenido de otro dato `string` que se va formando concatenando tres parejas de valores compuestas de dos espacios en blanco y un dígito.

    Nota: Más adelante aprenderemos a comprobar si el número introducido tiene tres dígitos, y cómo actuar (repitiendo la lectura) en el caso de que no los tenga.

    Dificultad Media.

27. Supongamos el siguiente código:

    ```
    int entero;
    char caracter;

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

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

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

    La variable `caracter` almacena internamente el valor 55 (número ASCII del carácter 7). La variable `entero` almacenará, también, el valor 55

    Queremos construir un programa para asignarle a la variable `entero` el *número* 7, asociado al dígito representado en la variable `caracter`, es decir, el valor 7 (no el 55) ¿Cómo lo haría? El programa debe mostrar el resultado.

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

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

28. Para intercambiar mensajes de forma privada, se utilizan distintos algoritmos que codifican/descodifican una cadena de caracteres. Uno de los más sencillos y que fue utilizado por Julio César durante la época del Imperio Romano es el de *rotación*. Consiste en seleccionar una clave (un número entero), y desplazar las letras del alfabeto tantas posiciones como indica la clave. Trabajaremos únicamente con mayúsculas.

    Se considera una representación circular del alfabeto, de tal forma que el carácter que sigue a `'Z'` es `'A'`. Por ejemplo, si `clave`=4, entonces la `'A'` se reemplaza por la `'E'` y la `'Z'` por la `'D'`. Utilizando `clave`=0 la secuencia cifrada es igual a la original.

    Construya un programa que lea un entero representando la clave y un carácter (supondremos que se introduce correctamente una letra mayúscula del alfabeto inglés). El programa codificará el carácter según la clave introducida y lo mostrará por pantalla.

    Recomendamos que revise la solución del ejercicio 22 de esta misma Relación de Problemas.

    Finalidad: Expresiones con caracteres y enteros. Dificultad Media.

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

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

30. 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`.

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

31. Indicar si se produce un problema de precisión o de desbordamiento en los siguientes ejemplos indicando cuál sería el resultado final de las operaciones.

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

e)
```
double real, otro;
real = 2e34;
otro = real + 1;
otro = otro - real;
```

b)
```
long grande;
int chico1, chico2;
chico1 = 123456789;
chico2 = 123456780;
grande = chico1 * chico2;
```

c)
```
double res, real1, real2;
real1 = 123.1;
real2 = 124.2;
res = real1 * real2;
```

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

f)
```
double real, otro;
real = 1e+300;
otro = 1e+200;
otro = otro * real;
```

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

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.

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

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

    Escribid una expresión lógica que sea verdadera si una variable de tipo entero llamada `edad` es mayor o igual que 18 y menor que 67.

    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 una expresión lógica que nos informe si el valor de una variable `double` llamada `distancia` es menor que la constante `LIMITE`.

    Usad una variable lógica que registre si el valor de una variable `int` es menor que otra, otra que informe si son iguales y otra que informe si es mayor. Asegúrese que sólo una, y sólo una de las tres trendrá el valor `true`.

    Nota: 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: Usar expresiones lógicas, muy usadas en el tema 2. Dificultad Baja.

33. 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.

34. 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:

    ```
    Grados Fahrenheit = (Grados Celsius * 180 / 100) + 32
    ```

    Buscad en Internet el por qué de dicha fórmula.

    Dificultad Baja.

35. 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:

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

    Se pide crear un programa que lea un entero positivo n, y calcule el mínimo número de dígitos que se necesitan para su representación. Para simplificar los cómputos, 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.

36. Se quiere construir un programa que lea un número real `r` y un número entero `n` y trunque `r` a tantas cifras decimales como indique `n`. El resultado debe guardarse en una variable diferente. La única función que puede usar de `cmath` es `pow`.

    Por ejemplo, si `r` vale `1.2349` y `n` vale 2 el resultado será `1.23`, si `r` vale `1.237` y `n` vale 2 el resultado será `1.23`, si `r` vale `1.237` y `n` vale 1 el resultado será `1.2` y si si `r` vale `1.237` y `n` vale 0 el resultado será `1`

37. Cread un programa que lea las coordenadas de dos puntos $\boldsymbol{P_1 = (x_1, y_1)}$ y $\boldsymbol{P_2 = (x_2, y_2)}$ y calcule la distancia euclídea entre ellos:

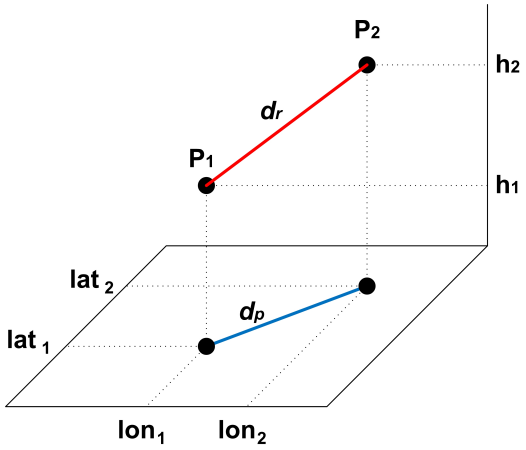
    $$\boldsymbol{d(P_1, P_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}}$$

    Dificultad Baja.

38. El sistema de posicionamiento global, más conocido por sus siglas en inglés, **GPS** (**G**lobal **P**ositioning **S**ystem), es un sistema que permite determinar en toda la Tierra la posición de un objeto.

Un dispositivo GPS es capaz de captar y registrar la posición en el espacio en base a tres coordenadas: **latitud** y **longitud** (*grados*) y **altura** (*metros*). Los valores de latitud y longitud deben verificar $-90 \leq \text{lat} \leq 90$ y $-180 < \text{lon} \leq 180$.

Construir un programa que lea la latitud y longitud (posición en el plano) de dos puntos y calcule la distancia *sobre plano* entre los dos puntos (distancia que no considera la altura de los puntos). Se trata de la longitud del segmento d_p en la figura.



Use la llamada *fórmula del Haversine*:

a) Calcular a:

$$a = \sin^2\left(\frac{1}{2}(\text{lat}_2 - \text{lat}_1)\right) + \cos(\text{lat}_1)\cos(\text{lat}_2)\sin^2\left(\frac{1}{2}(\text{lon}_2 - \text{lon}_1)\right)$$

b) Calcular $c = 2\arcsin(\text{mín}(1, \sqrt{a}))$

c) La distancia será $d_p = R\,c$
donde $R = 6372797{,}560856$ m es la longitud media del radio terrestre.

Tenga en cuenta lo siguiente:

a) Los datos de latitud y longitud en las fórmulas vienen expresados en *radianes*. Recuerde que los valores de latitud y longitud que se leen están expresados en grados por lo que habrá de transformarlos a radianes.

b) La función arco-seno ($\arcsin$) viene ya implementada en la biblioteca `cmath` con el nombre `asin`

c) mín representa el mínimo de dos valores. Para implementarlo utilice la función `min` que viene definida en la biblioteca `algorithm`.

Una vez calculada la distancia sobre plano, calcular la distancia *real* entre los dos puntos (que considera sus alturas). Se trata de la longitud del segmento d_r de la figura. Observe que puede formar un triángulo rectángulo a partir de los segmentos d_p, d_r y la diferencia de las alturas.

Dificultad Media.

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

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

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

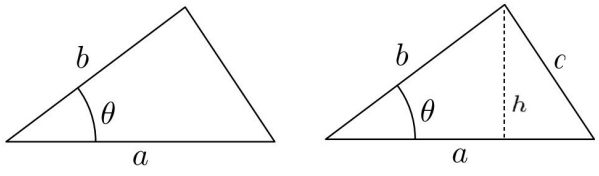
    c) $\sqrt{1+\left(\dfrac{e^x}{x^2}\right)^2}$

Algunas funciones de `cmath`
$\mathrm{sen}(x) \longrightarrow$ `sin(x)`
$\cos(x) \longrightarrow$ `cos(x)`
$x^y \longrightarrow$ `pow(x, y)`
$\ln(x) \longrightarrow$ `log(x)`
$\mathrm{e}^x \longrightarrow$ `exp(x)`

    Dificultad Baja.

40. 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 (figura A) 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 (tenga en cuenta que el argumento de la función `sin` va en radianes).



A B

    Dificultad Baja.

    Demostración: Nombramos a los lados del triángulo a, b y c, y consideramos que h es la altura sobre el lado a. Consideramos únicamente el ángulo θ delimitado por los lados a y b (figura B). Por la definición de seno, $sin(\theta) = h/b$, o sea, $h = b\,sin(\theta)$.

    Aplicando la fórmula del área del triángulo tenemos que $A = \frac{1}{2}a\,h$ y sustituyendo en la fórmula del área el valor de h tenemos que $A = \frac{1}{2}\,a\,b\,sin(\theta)$

41. Suponga un caso de elección en el que se decide entre tres opciones: **A**, **B** y **C**.

    - El número de votantes totales (censo) está registrado en la constante `VOTANTES`.
    - El número total de de votos registrados es `votos_emitidos`, donde el número total de de votos presenciales es `votos_presenciales` y el número total de de votos por correo es `votos_correo`.

- Los votos registrados se dividen en válidos (`votos_validos`) y nulos (`votos_nulos`).
- Entre los votos válidos se distinguen entre los recibidos por correo (`votos_validos_correo`) y presencialmente (`votos_validos_presenciales`).
- Cada opción recibe un número de votos válidos que se registra en las variables `votos_A`, `votos_B` y `votos_C`, respectivamente.

Escriba expresiones lógicas para reflejar las situaciones que se describen a continuación. Para cada caso, el resultado debe guardarse en una variable lógica (escoja un nombre adecuado).

a) El número de votos emitidos totales es mayor que el número de votantes.

b) El número de votos válidos no es igual a la suma de los votos recibidos por las tres opciones.

c) El número de votos registrados es igual a la suma de los votos válidos y nulos.

d) Gana la opción **A**.

e) Gana la opción **B**.

f) Gana la opción **C**.

g) Empatan dos opciones.

h) Hay un *empate técnico* entre las opciones **A** y **B**. Definimos que ocurre un empate técnico cuando la diferencia entre los dos valores es menor que el 5 % de su suma.

i) La opción **A** obtiene mayoría absoluta.

j) Hay una coalición de dos opciones que obtiene mayoría absoluta.

k) La coalición **A**-**C** obtiene mayoría absoluta.

l) La opción **B** obtiene menos de 3 % de los votos válidos registrados.

m) La participación es mayor que el 75 % del censo de votantes.

n) Queremos saber si la opción **A** puede gobernar "sólo o en compañía de otros".

ñ) El número de votos nulos ó el número de abstenciones es mayor que el número de votos válidos.

o) El número de votos nulos y el número de abstenciones es mayor que el número de votos válidos, pero las abstenciones no son mayores que los votos nulos.

p) El número de votos por correo es mayor que el 20 % de los votos presenciales o el número de abstenciones.

q) El número de votos nulos por correo es mayor que el número de votos nulos presenciales.

r) El número de votos nulos por correo es mayor que el número de votos válidos por correo.

RELACIÓN DE PROBLEMAS II. Estructuras de Control

Ejercicios sobre condicionales

1. 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 750 euros y la persona es mayor de 55 años.

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

   Imprimid el resultado por pantalla. Si no procede la subida imprimid el mensaje `"No es aplicable la subida"`. En ambos casos imprimid el salario final.

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

2. Realizar un programa que lea dos valores enteros desde teclado y calcule si cualquiera de ellos divide o no (de forma entera) al otro.

   - En el caso que uno de ellos divida al otro deberá indicarnos que se ha detectado que uno de ellos es múltiplo del otro, y cuál es.
   - Si no fueran múltiplos lo indicará.

   Debemos contemplar el caso en el que alguno de los valores introducidos sea cero, en cuyo caso, ninguno divide al otro.

   Tened especial cuidado en no mezclar entradas/salidas con cálculos.

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

3. Escribid un programa 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 doble con una expresión lógica compuesta. Dificultad Baja.

4. Retome la solución del ejercicio 7 de la *Relación de Problemas I*. La forma de calcular la tarifa final del billete cambia ahora de la forma siguiente: la tarifa base sigue siendo de 150 euros, la misma para todos los destinos. Ahora bien, si el destino está a menos de 300 kilómetros, el precio final es la tarifa base. Para destinos a más de 300 Km, se suman 10 céntimos por cada kilómetro de distancia al destino (a partir del Km 300).

Cree un programa para que lea el número de kilómetros al destino y calcule el precio final del billete.

Finalidad: Plantear una estructura condicional simple. Actualizar una variable según una condición. *Dificultad Baja.*

5. Continuando con el problema del cálculo de las tarifas aereas (ejercicio 4) ahora se van a aplicar descuentos a las tarifas ya calculadas. Los criterios a aplicar son los siguientes:

   a) Un primer descuento está determinado por el número de kilómetros del trayecto. Si el trayecto es mayor de 700 km, se aplica un descuento del 2 %

   b) El segundo descuento dependerá del número de puntos de la tarjeta de fidelización del cliente.
   Si el número de puntos es mayor de 100, se aplica un descuento del 3 % y si es mayor de 200, se aplica un descuento del 4 %.

   Los dos descuentos anteriores son independientes y acumulables. En cualquier caso, ambos se aplican sobre el precio del billete: en este programa un cliente podría beneficiarse de un descuento del 2 %, del 3 %, del 4 %, del 5 %, del 6 %, o ninguno.

   Finalidad: Plantear una estructura condicional simple. *Dificultad Media.*

6. Ampliad el ejercicio 19 de la *Relación de Problemas I* de manera que los dos instantes puedan pertenecer a dos días consecutivos. Si los dos instantes pertenecen a dos días diferentes, indicadlo de manera explícita en la salida.

   Calcule la solución usando dos algoritmos: 1) en base al número de segundos transcurridos entre los dos instantes, y 2) comparando los valores de horas, minutos y segundos de los dos instantes.

   Escribid las dos soluciones en el mismo programa.

   Finalidad: Trabajar con condicionales. *Dificultad Media.*

7. 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.

   Nota: El programa sólo procesará años del siglo XX en adelante.

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

8. Modifique la solución del ejercicio 38 de la *Relación de Problemas I* de manera que el mínimo:
   ...
   c) Calcular $c = 2\ \arcsin(\text{mín}(1, \sqrt{a}))$

...

que se calculaba con la función `min` de la biblioteca `algorithm` deberá calcularse ahora con un condicional.

Finalidad: Plantear una estructura condicional simple. Dificultad Baja.

9. Modifique la solución del ejercicio 9 de la *Relación de Problemas I* y cambie la implementación para realizar la codificación del carácter utilizando un condicional simple en vez de una expresión tan compleja como la que se utilizó en dicho ejercicio.

   En definitiva, en este ejercicio **no** puede usar el operador módulo `%`, aunque sí puede usar condicionales simples.

   Finalidad: Plantear una estructura condicional simple. Actualizar una variable según una condición. Dificultad Baja.

10. Los radares (o cinemómetros) pueden ser fijos o móviles tienen como misión registrar la velocidad de los vehículos. Estos aparatos presentan un *margen de error* que hay que aplicar a la velocidad captada por el radar y vienen especificados en la orden ITC/3123/2010, de 26 de noviembre https://www.boe.es/diario_boe/txt.php?id=BOE-A-2010-18556

    a) En el caso de un radar fijo:
    - I) Si la velocidad captada por el radar es menor o igual que 100 km/h, el margen de error es de 5 km/h.
    - II) En caso contrario, el margen de error es de un 5 %

    b) En el caso de un radar móvil:
    - I) Si la velocidad captada por el radar es menor o igual que 100 km/h, el margen de error es de 7 km/h.
    - II) En caso contrario, el margen de error es de un 7 %

    Los márgenes de error se aplican sobre la velocidad captada y da como resultado la *velocidad imputada*. Por ejemplo, si la velocidad captada es de 95 km/h, la velocidad imputada sería de 95 - 5 = 90 km/h en el caso de un radar fijo y de 95 - 7 = 88 km/h en el caso de un radar móvil.

    Si la velocidad captada es, por ejemplo, de 104 km/h, la velocidad imputada sería de `104 - 5% de 104 = 104 - 5.2 = 98.8 km/h` en el caso de un radar fijo y de `104 - 7% de 104 = 104 - 7.28 = 96.72 km/h` en el caso de un radar móvil.

    Construya un programa que lea desde teclado un carácter que indique el tipo de radar (`F` para fijo y cualquier otra letra para móvil), la velocidad captada, e imprima la velocidad imputada.

    Finalidad: Plantear una estructura condicional anidada y actualizar variables en función de ciertas condiciones. Dificultad Baja.

11. La Dirección General de Tráfico publica en su página web las sanciones a aplicar por infracción de velocidad. Puede consultarse la tabla en https://sede.dgt.gob.es/Galerias/tramites-y-multas/alguna-multa/consulta-de-sanciones-por-exceso-velocidad/cuadro_velocidad.pdf

    En este ejercicio queremos determinar la sanción a aplicar en una autovía, cuyo límite de velocidad es 120 Km/h. En la siguiente tabla se muestra la velocidad del vehículo y la sanción correspondiente (número de puntos del carnet de conducir que se restan y la multa en euros)

    ```
    (120,150] -> 0, 100
    (150,170] -> 2, 300
    (170,180] -> 4, 400
    (180,190] -> 6, 500
    (190, oo) -> 6, 600
    ```

    Escriba un programa que lea la velocidad del vehículo e imprima en pantalla la sanción correspondiente (número de puntos a detraer y multa en euros)

    Finalidad: Planteamiento de una estructura condicional anidada. Dificultad Baja.

12. Ampliad el ejercicio 21 de la *Relación de Problemas I*. Después de haber leído los datos que definen el intervalo, el programa debe leer un valor real y determinar si está o no dentro del intervalo.

    Finalidad: Plantear una condición compleja. Dificultad Media.

13. Se quiere leer un carácter `letra_original` desde teclado y

    - 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`

    Finalmente, imprimiremos 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.`

    Finalidad: Plantear una estructura condicional anidada. Diseñar programas que separen entradas/salidas y cálculos. Dificultad Baja.

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

    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.*

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

    - 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:
      Renta_neta = Renta_bruta - Renta_bruta * Retención / 100

    Los datos a leer son:

    - Si la persona es un trabajador autónomo o no.
    - Si es pensionista o no.
    - Si está casado o no.
    - Renta bruta (total de ingresos obtenidos).
    - Retención inicial.

    La modificación se hará de la siguiente forma:

    - Bajar un 3 % la retención fiscal a los autónomos
    - Para los no autónomos:
      - Se sube un 1 % la retención fiscal a todos los pensionistas
      - Al resto de los trabajadores se les aplica una subida lineal del 2 %.
        Una vez hecha esta subida, se le aplica (sobre el resultado anterior) las siguientes subidas adicionales a la retención fiscal:
        - Subir otro 2 % si la renta bruta es menor de 20.000 €
        - Subir otro 2.5 % a los casados con renta bruta superior a 20.000 €
        - Subir otro 3 % a los solteros con renta bruta superior a 20.000 €

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.*

16. La tabla para el cálculo del precio a pagar en un *parking* para 2020 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 120: 0.0311 euros cada minuto
    Desde el minuto 121 al 660: 0.0305 euros cada minuto
    Desde el minuto 661 hasta máximo 24 horas: 24.00 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 94 minutos, los primeros 30 minutos se facturan a 0.0412 el minuto, los siguientes 60 se facturan a 0.0370 el minuto, y los 4 restantes a 0.0311 el minuto.

    Nota: Se supone que la estancia no sobrepasa las 24 horas.

    Finalidad: Utilización del condicional doble. *Dificultad Media.*

17. Se acuerda la actualización de las tarifas de aparcamiento incorporando un nuevo tramo (desde el minuto 661 hasta el 900 a un coste de 0.0270 euros cada minuto) y subiendo el coste de la tarifa plana a 35 euros (aplicable ahora desde el minuto 901).

    Actualice el programa con estas nuevas condiciones. Hágalo sobre la solución entregada y sobre la solución propuesta por el profesor.

    Finalidad: Evaluar informalmente el coste de las actualizaciones. *Dificultad Baja.*

18. Realizar un programa que realice la conversión entre dos unidades de distancia del SI. Las unidades contempladas serán: `mm` (milímetros), `cm` (centímetros), `m` (metros) y `km` (kilómetros).

    El programa mostrará el *prompt* > y el usuario escribirá cuatro *tokens*:

    valor unidad_inicial `a` *unidad_final*

    y el programa mostrará el resultado de la transformación. Por ejemplo, si quisiéramos transformar 7 centímetros a metros escribiremos:

```
> 7 cm a m
7.000 cm  = 0.007 m
```

    y para convertir 5.2 kilómetros a milímetros:

```
> 5.2 km a mm
5.200 km  = 5200000.000 mm
```

Notas:

a) Se permite la transformación entre todas las unidades citadas anteriormente.

b) Intente minimizar el número de comprobaciones a realizar.

c) Tenga en cuenta todas las situaciones de error. En caso de error deberá informar del error y terminar la ejecución del programa.

Finalidad: Gestionar múltiples situaciones que resultan de evaluar expresiones lógicas y minimizar el número de comprobaciones. Dificultad Media.

Ejercicios sobre tipos enumerados

19. Modificad el ejercicio 3 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*.

20. 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 %, mientras que si su salario es mayor o igual que 300 euros pero inferior a 900 euros se le subirá un sólo el 1.5 %.

    Imprimid el resultado por pantalla indicando qué subidas se han aplicado.

    Finalidad: Plantear una estructura condicional anidada. Usar el tipo enumerado para gestionar categorías. *Dificultad Baja.*

21. Tome como referencia la solución al ejercicio 13 de esta misma relación de problemas. Ahora deberá emplear un tipo enumerado que permita guardar el *tipo* de carácter de `letra_original`. Considere que ahora podría ser: una letra mayúscula, una letra minúscula, un dígito u otro carácter.

    Importante: las vocales con tilde (mayúsculas y minúsculas) y las letras `ú`, `Ü`, `ñ` y `Ñ` se englobarán en la categoría de *otros* por simplificar la solución.

    Finalidad: Separación de E/S y C. Usar el tipo enumerado para gestionar categorías. *Dificultad Media.*

Ejercicios sobre bucles

22. 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 (dicho de otra manera, habrá que **filtrar** el valor leído para que sólo admita letras mayúsculas).

    Calculad la minúscula correspondiente e imprimidla en pantalla.

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

23. Ampliad el ejercicio 6 de manera que en esta versión filtre adecuadamente los valores de las horas, minutos y segundos.

    Nota: Para calcular si los dos instantes pertenecen al mismo día emplee el algoritmo 2 (comparación entre horas, minutos y segundos de los dos instantes).

    Finalidad: Trabajar con filtros. Dificultad Baja.

24. Ampliad el ejercicio 24 de manera que en esta versión se filtre adecuadamente el valor de la desviación típica.

    Finalidad: Trabajar con filtros. Dificultad Baja.

25. Reescribid la solución del problema del cálculo de las tarifas aereas con descuestos (ejercicio 5). Ahora se filtrarán adecuadamente los datos de entrada (distancia a recorrer en el vuelo y puntos del cliente).

    Finalidad: Trabajar con filtros. Dificultad Baja.

26. Realizar un programa que lea desde teclado un entero positivo 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.

    Finalidad: Bucle sencillo y filtro de entrada de datos. Dificultad Baja.

27. Ampliad el ejercicio 26 de la *Relación de Problemas I* de manera que en esta nueva versión lea un número entero positivo en una dato `int` o `string` comprobando la validez del dato leído (debe ser positivo).

    Por ejemplo, si el número (`int`) es 3519, la salida sería `|3|5|1|9|`

    En este ejercicio puede mezclar entradas, cálculos y salidas.

    Finalidad: Practicar con filtros y ciclos básicos. Dificultad Baja.

28. Modificad el ejercicio 27 de esta misma relación de problemas para que ahora estén *claramente diferenciadas las etapas de entrada, cálculos y salidas*. Para ello componga un `string` con los dígitos del número leído separados por una barra vertical.

    Por ejemplo, si el número (`int`) es 3519, la salida (`string`) sería `|3|5|1|9|`

    Finalidad: Practicar con filtros y ciclos básicos. Dificultad Media.

29. Modifiquemos el ejercicio 10 de la *Relación de Problemas I*. Supongamos ahora que se quiere reinvertir todo el dinero obtenido (capital más intereses) en otro plazo fijo a un año. Y así, sucesivamente. Construid un programa para que lea el capital C, el interés I y un número de años N, y calcule e imprima todo el dinero obtenido durante cada uno de los N años, suponiendo que se reinvierte todo.

    Filtrar adecuadamente los valores leidos de manera que cumplan las condiciones:

    - Sobre el capital inicial, $C > 0$,
    - Sobre el número de años, $1 <= N <= 20$
    - Sobre el interés, $0 < I <= 10$

    El programa debe mostrar una salida del tipo:

    ```
    Total en el año número 1 = 240
    Total en el año número 2 = 288
    ......................
    ```

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

    Finalidad: Practicar con filtros y ciclos básicos. Dificultad Baja.

30. Modifiquemos nuevamente el ejercicio 10 de la *Relación de Problemas I*. Ahora se trata de construir un programa para calcular cuantos años han de pasar hasta llegar a doblar, como mínimo, el capital inicial.

    Los valores del capital inicial y del interés deben cumplir las restricciones indicadas en el ejercicio 29.

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

31. Usando como base el programa que soluciona el ejercicio 17 escribid un programa que lea la hora actual (minutos y segundos, por separado) y una cantidad de dinero (en euros) y nos indique hasta qué hora podemos tener el coche aparcado si vamos a pagar con la cantidad especificada.

    Implementar los filtros adecuados al problema.

    Si dispone de 5 euros y 25 céntimos deberá indicar el valor `5.25` cuando se le pida el dinero disponible.

    Finalidad: Practicar con filtros y ciclos básicos. Dificultad Baja.

32. De http://countrymeters.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 personada cada 71.9 segundos

    Escribid un programa que pida un valor de población y calcule cuántos años (enteros) transcurrirán hasta que la población estimada sea mayor o igual al valor dado.

    Nota: La población en actual (Octubre de 2019) es de 1.406.862.401 personas,

    Filtrar el valor de población introducido para que sea mayor que el de la población actual.

    Finalidad: Practicar con filtros y ciclos básicos. Dificultad Baja.

33. Se dice que un número n es **triangular** si se expresa como la suma de los primeros k valores enteros. Por ejemplo, $n = 6$ es triangular ya que $6 = 1 + 2 + 3$ ($k = 3$).

    Una forma de obtener los números triangulares es a través de la fórmula $k(k+1)/2 \ \forall k \in \mathbb{N}$.

    Escriba un programa que, sin aplicar directamente la fórmula anterior, muestre todos los números triangulares que hay menores que un entero n introducido desde teclado.

    Finalidad: Practicar con filtros y ciclos básicos. Dificultad Baja.

34. Use como referencia el ejercicio 12 de esta misma relación de problemas.

    Después de haber leído los datos que definen el intervalo, el programa debe leer el valor de un entero n y a continuación debe leer n valores reales.

    Por cada uno de ellos, el programa nos dirá si pertenece o no al intervalo anterior.

    Finalidad: Plantear una estructura repetitiva básica para llevar un contador de acciones. Dificultad Baja.

35. Ampliad el ejercicio 7. El programa pedirá los valores de *dos años del siglo XXI* 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.

36. Realizar un programa que lea números reales **positivos** desde teclado y calcule cuántos se han introducido y cuál es el mínimo y el máximo de dichos valores. Se dejará de leer datos cuando el usuario introduzca el valor 0. Realizad la lectura de los reales dentro de un bucle sobre una única variable llamada `dato`. Es importante controlar los casos extremos, como por ejemplo, que el primer valor leído fuese ya el terminador de entrada (en este caso, el cero).

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

37. Realizar un programa que lea dos secuencias de números reales positivos y nos diga si todos los valores de la primera secuencia son mayores que todos los de la segunda.

    Realizad la lectura de los números dentro de sendos bucles sobre una única variable llamada `dato`. El final de cada secuencia viene marcado por el valor 0.

    Finalidad: Ejercitar el uso de bucles. *Dificultad Baja*.

38. Recupere la solución del ejercicio **??** de esta relación de problemas (cálculo de la velocidad imputada). Modifique el programa para que vaya leyendo los siguientes datos dentro de un bucle: en primer lugar, se leerá la matrícula del vehículo (en un dato de tipo `string`), a continuación el tipo de radar (dato de tipo `char`) y finalmente la velocidad del vehículo (en un dato de tipo `double`).

    Supondremos que el carácter `'F'` está asociado al radar fijo y el carácter `'M'` al radar móvil. Supondremos que el usuario introduce un carácter correcto.

    El programa calculará las velocidades imputadas e imprimirá en pantalla la matrícula de aquel vehículo que tenga máxima velocidad imputada (también imprimirá dicha velocidad). La lectura terminará cuando la matrícula del vehículo sea la cadena `"FIN"`

    Ejemplo de entrada: `4312BHG F 95 8342DFW F 104 4598IJG M 95 FIN`

    Salida correcta: `8342DFW 98.8`

    Finalidad: Practicar con ciclos de lectura adelantada. *Dificultad Baja.*

39. Una empresa que tiene tres sucursales decide llevar la contabilidad de las ventas de sus productos a lo largo de una semana. Registra cada venta con tres datos: 1) el identificador de la sucursal (1,2 ó 3), 2) el código del producto (`a`, `b` ó `c`) y 3) 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 lectura termina cuando la sucursal introducida vale -1.

    Por ejemplo, con la serie de datos que se detalla a continuación:

```
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
```

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 redirigid la entrada desde 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.*

40. Se decide informatizar el acta de un partido de baloncesto para saber qué equipo es el ganador del partido. El acta de anotaciones está formada por una serie de *tríos* de números, y cada uno contiene: el número de equipo (1 ó 2), el dorsal del jugador y el número de puntos conseguidos. La última anotación es el valor terminador -1.

    Por ejemplo, con la entrada

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

    El programa muestra el equipo ganador (o si hay empate) y la puntuación.

    Por ejemplo, con la entrada anterior, gana el equipo 1, y la puntuación es 10-7.

    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.*

41. 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.

    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.*

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

    Por ejemplo, ante la entrada siguiente:

    ```
    17.2 17.3 16.2 16.4 17.1 19.2 18.9 -1
    ```

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

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

Finalidad: Trabajar con bucles en los que se compara un valor leído con otro leído en la iteración anterior. Dificultad Media.

43. Modifique el ejercicio 42 para que la lectura de datos finalice cuando se introduce un valor fuera de un rango establecido entre los valores `MIN_RANGO` y `MAX_RANGO`.

44. Para controlar los tiempos que emplean los corredores de una carrera popular se desea disponer de un programa muy sencillo que se irá ampliando en sucesivas versiones. En esta primera versión el programa empezará pidiendo la hora de inicio de la carrera (hora, minuto y segundo) y la distancia.

    Después, para cada corredor que llega a meta (por orden de llegada) se pide el número de dorsal y la hora de llegada (hora, minuto y segundo). Tras leer estos datos el programa mostrará el tiempo que ha empleado en completar la carrera (horas, minutos y segundos) y su ritmo (en km/h y en minutos/km).

    El programa finalizará cuando al solicitar el número de dorsal se introduzca un valor negativo.

    Nota: Deberá filtrar **todos** los datos leídos.

    Finalidad: Trabajar con bucles de lectura anticipada. Trabajar con bucles en los que se compara un valor leído con otro leído en la iteración anterior. Dificultad Media.

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

    Realizar un programa que lea una secuencia de números naturales terminada con un número negativo y la codifique mediante el método RLE. Leed los datos desde un fichero externo, tal y como se explica en la página 24.

Entrada:	1 1 1 2 2 2 2 2 3 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.

46. Implemente un programa que lea bits (ceros y unos) desde teclado, hasta que se introduzca un valor negativo. Si se introduce un positivo distinto de 0 y 1, el programa lo descartará y volverá a leer un valor.

Cada 8 valores de bits leídos, el programa calculará el número entero que representa y lo transformará en el carácter (`char`) correspondiente. Debe tener en cuenta que el primer bit leído es el más significativo.

Por ejemplo, si el usuario introduce los siguientes enteros:

```
0 1 0 0 1 3 4 1 1 1
```

acaba de completar un octeto ya que el 3 y el 4 no son bits (0 o 1) y se descartan. Por tanto, el octeto completado es 0 1 0 0 1 1 1 1 que corresponde al número entero 79 y por tanto al carácter `'O'`. El programa debe ir construyendo el entero 79 como suma de potencias de 2 ($0 \cdot 2^7 + 1 \cdot 2^6 + \ldots + 1 \cdot 2^0$)

Si no es posible completar el último bloque con 8 bits (porque se haya introducido un negativo antes del octavo bit), se descartarán todos los bits de ese último bloque incompleto.

Si el carácter obtenido corresponde a una letra -mayúscula o minúscula- lo mostrará por pantalla. Una vez terminada la entrada de datos, el programa mostrará el porcentaje de letras y otros símbolos (distintos de letras) leídos.

Ejemplo de entrada:

```
0 1 0 0 1 3 4 1 1 1 2 0 1 1 0 1 0 1 1 9 0 0 1 7 0 0 0 0 1 1 0 -1
```

Salida correcta:

```
Ok
Letras: 66.67%
Otros: 33.33%
```

Donde las correspondencias son `01001111` (carácter `O`), `01101011` (carácter `k`), y `00100001` (carácter `!`). Los bits finales 10 se descartan ya que no se ha completado un bloque de ocho.

Finalidad: Bucles anidados y lectura de datos. Dificultad Media.

47. Escribir un programa que lea un número entero positivo y muestre su descomposición en factores primos.

    Hágalo de manera que se calcule y muestre el resultado de dos maneras:

    a) La primera (más sencilla) consiste en enumerar todos los primos de la descomposición (en el caso de que un primo intervenga más de una vez se mostrará tantas veces como sea preciso). Por ejemplo:

    ```
    360 = 2 * 2 * 2 * 3 * 3 * 5
    121 = 11 * 11
    11 = 11
    ```

b) La segunda consiste en expresar el resultado como el producto de potencias (en el caso de que un primo intervenga más de una vez se mostrará una potencia cuya base sea el valor del primo y el exponente será el número de veces que se repite). Evite potencias triviales (de exponente 1) escribiendo, en ese caso, únicamente la base (el valor del primo). Por ejemplo:

```
360 = 2^3 * 3^2 * 5
121 = 11^2
11 = 11
```

48. El algoritmo de la multiplicación rusa es una forma distinta de calcular la multiplicación de dos números enteros $n * m$. Para ello este algoritmo va multiplicando por 2 el multiplicador m y dividiendo (sin decimales) por dos el multiplicando n hasta que n tome el valor 1 y suma todos aquellos multiplicadores cuyos multiplicandos sean impares. Por ejemplo, para multiplicar 37 y 12 se harían las siguientes iteraciones

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

El resultado de multiplicar 37 y 12 sería la suma de los multiplicadores correspondientes a los multiplicandos impares (en negrita), es decir 12+48+384=444

Cread un programa que lea dos enteros y calcule su producto con este algoritmo.

Dificultad Media.

49. Diseñar un programa para jugar a adivinar un número. En cada jugada el jugador introducirá un valor y el el juego indicará 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 no quiera seguir jugando a adivinar ese número (usad un valor especial -terminador- para esta opción).

Dificultad Media.

Para poder generar números aleatorios en un rango determinado por `MIN` y `MAX` será necesario incluir las siguientes instrucciones:

```
...
#include<ctime>
#include<cstdlib>
...
const int MIN =   1;
const int MAX = 100;
```

```
const NUM_VALORES = MAX-MIN + 1; // Número de valores posibles

time_t t;
srand(time(&t)); // Inicializa el generador de núms. aleatorios
                 // con el reloj del sistema (hora actual)
int incognita; // número aleatoria a generar

// Generación de incognita (MIN <= incognita <= MAX)
incognita = (rand() % NUM_VALORES) + MIN;
```

La orden `srand(time(&t))` debe ejecutarse una única vez al principio del programa y sirve para inicializar la secuencia de números aleatorios. Posteriormente, cada vez que se ejecute:

```
incognita = (rand() % NUM_VALORES) + MIN;
```

la instrucción `rand` devolverá valor (pseudo)aleatorio que se modificará para que quede comprendido entre `MIN` y `MAX`, y finalmente asignado a la variable `incognita`.

50. Recupere la solución del ejercicio 24 (función gaussiana) de la *Relación de Problemas I*. Construir un programa para imprimir el resultado de aplicar dicha función a varios valores de abscisas.

    En primer lugar, se leerán los parámetros que definen la función, es decir, la *esperanza* μ y la *desviación típica* σ. La esperanza puede ser cualquier valor, pero la desviación típica debe ser mayor o igual que cero.

    A continuación el programa pedirá un valor *mínimo* (x_i), un valor *máximo* (x_f) y un *incremento* (Δ). Filtrar adecuadamente estos valores.

    El programa mostrará el valor de la función gaussiana (g en la ecuación 1) para todos los valores de x (la abscisa) entre x_i y x_f considerando entre dos puntos consecutivos un salto igual al valor dado en el incremento Δ.

    En definitiva, se trata de calcular $g(x)$ si $x = x_i + k\,\Delta$ $(k = 0, 1, 2, \ldots)$ y $x \leq x_f$

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

    En la figura 10.A mostramos cuatro funciones gaussianas con diferentes parámetros.

    Finalidad: usar cógido verificado en bucles sencillos. Dificultad Baja.

51. Ahora estamos interesados en obtener el área que cubre la función gaussiana en un un intervalo dado. Si consideramos el intervalo $[-\infty, x]$ el valor del área de la

región que queda bajo la curva se calcula con la *distribución acumulada (cumulative distribution function)* en el punto x, abreviado $CDF(x)$. Matemáticamente:

$$CDF(x) = \int_{-\infty}^{x} g(x)\,dx$$

donde $g(x)$ es el valor de la función gaussiana en x (ecuación 1).

Puede obtenerse un valor aproximado de $CDF(x)$ como la *suma* de valores de $g(x)$, empezando por valores alejados de la media (y menores que ella).

En la figura 10 mostramos cuatro funciones gaussianas con diferentes parámetros (A) y sus correspondientes CDFs (B). En la figura 10.A mostramos el área bajo la curva de $g(x)$ con μ =2.0 y σ^2 =0.5 en el intervalo $[-\infty$,-2.5] y su valor es CDF(-2.5).

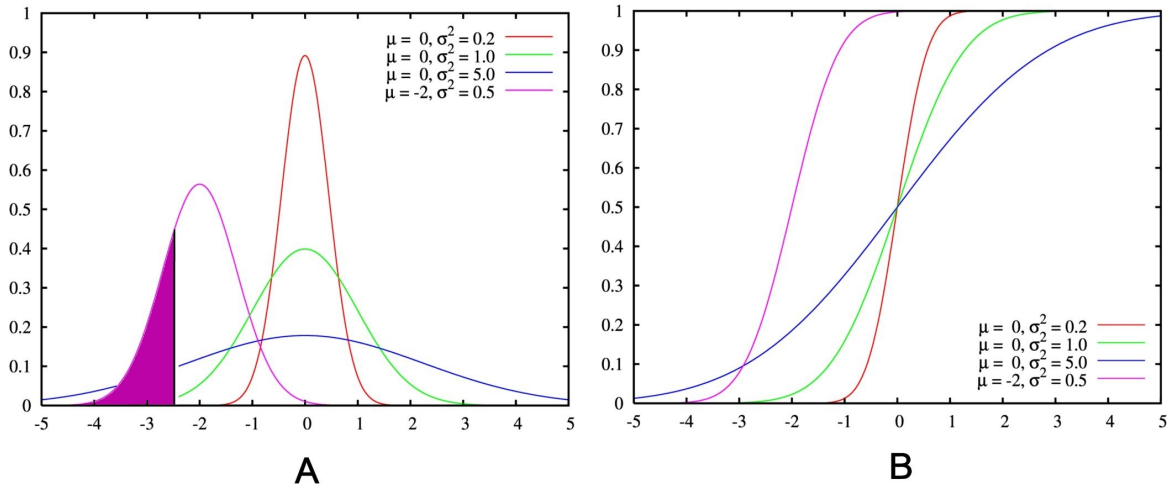


Figura 10: Distribuciones gaussianas y sus correspondientes CDFs
Fuente: *Wikipedia: Distribución normal*
https://es.wikipedia.org/w/index.php?title=Distribucion_normal

Escribid un programa calcule $CDF(x)$ para un valor de x dado. El programa debe pedir los parámetros que definen una función gaussiana (μ y σ) y el valor de la abscisa, x, para el que se va a calcular $CDF(x)$.

Para el cálculo práctico de $CDF(x)$ la integral se convierte en una **suma**, y se requiere concretar cuántas sumas se van a realizar. Bastará con indicar:

a) un valor inicial para x (tómese $\mu - 3\sigma$).

b) un "salto" entre dos valores consecutivos de x (use una constante).

Puede usar la calculadora de:
https://solvemymath.com/online_math_calculator/statistics/cdf_calculator.php para comprobar su resultado.

Dificultad Media.

52. En el siglo XIV el matemático indio Madhava de Sangamagrama calculó el arco tangente a través de un desarrollo de Taylor:

$$\mathbf{arctan}(x) \;=\; \sum_{i=0}^{\infty} \frac{(-1)^i \, x^{2i+1}}{2i+1}$$

Usando como x el valor $\dfrac{1}{\sqrt{3}}$ obtenemos:

$$\mathbf{arctan}(\frac{1}{\sqrt{3}}) \;=\; \frac{\pi}{6} \;=\; \sum_{i=0}^{\infty} \frac{(-1)^i \, (\frac{1}{\sqrt{3}})^{2i+1}}{2i+1}$$

Por lo tanto, podemos usar la siguiente aproximación:

$$\frac{\pi}{6} \;=\; \sum_{i=0}^{\texttt{tope}} \frac{(-1)^i \, (\frac{1}{\sqrt{3}})^{2i+1}}{2i+1}$$

Construya un programa que:

a) lea el valor `tope` obligando a que esté entre 1 y 100000, calcule la aproximación de π mediante la anterior serie e imprima el resultado en pantalla.

b) lea un entero que indique cuántos decimales de precisión deseamos, calcule la aproximación de π mediante la anterior serie y nos indique cuántas iteraciones -sumas- ha tenido que realizar para conseguir la aproximación deseada.
El valor de π a aproximar es `3.14159265358979323846`

Importante: En la implementación de esta solución: 1) **NO** puede usar `pow` (observe que $(-1)^i$ es 1 cuando i es par y -1 si es impar) y 2) tampoco el condicional `if`.

Finalidad: Bucles simples. Acumulación de sumas. Comparación de reales. Dificultad Baja.

53. En el siglo XVII el matemático alemán Gottfried Leibniz y el matemático escocés James Gregory introdujeron una forma de calcular π a través de una serie, es decir, de una suma de términos:

$$\frac{\pi}{4} = \sum_{i=0}^{\infty} \frac{(-1)^i}{2i+1} = 1 - \frac{1}{2*1+1} + \frac{1}{2*2+1} - \frac{1}{2*3+1} + \cdots$$

Esta es una serie infinita, pues realiza la suma de infinitos términos. Como no podemos realizar un número infinito de operaciones, habrá que parar en un índice dado,

llamémosle `tope`, obteniendo por tanto una aproximación al valor de π. Usaremos el símbolo $\approx$ para denotar esta aproximación:

$$\frac{\pi}{4} \approx \sum_{i=0}^{i=\texttt{tope}} \frac{(-1)^i}{2i+1} = 1 - \frac{1}{2*1+1} + \frac{1}{2*2+1} - \frac{1}{2*3+1} + \cdots + \frac{(-1)^{\texttt{tope}}}{2*\texttt{tope}+1}$$

$$= 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots + \frac{(-1)^{\texttt{tope}}}{2*\texttt{tope}+1}$$

Construya un programa que lea el valor `tope` obligando a que esté entre 1 y 100000, calcule la aproximación de π mediante la anterior serie e imprima el resultado.

Resuelva este problema de dos formas distintas:

a) Usando la función `pow` de `cmath` para implementar $(-1)^i$

b) Sin usar la función `pow`. Observe que el valor de $(-1)^i$ es 1 para los valores pares de i y -1 para los impares.

Finalidad: Bucles simples. Acumulación de sumas. Dificultad Baja.

54. Otra aproximación de π introducida en el siglo XVII por el matemático inglés John Wallis viene dada por:

$$\frac{\pi}{2} \approx \frac{2}{1} \cdot \frac{2}{3} \cdot \frac{4}{3} \cdot \frac{4}{5} \cdot \frac{6}{5} \cdot \frac{6}{7} \cdot \cdots$$

Construya un programa que lea el valor `tope` obligando a que esté entre 1 y 100000, calcule la aproximación de π mediante la anterior fórmula (multiplicando un total de `tope` fracciones) e imprima el resultado en pantalla.

Debe resolver este problema de dos formas distintas, a saber:

a) Observe que el numerador y el denominador varían de forma alternativa (aunque ambos de la misma forma, a saltos de 2). Cuando a uno le toca cambiar, el otro permanece igual. Este comportamiento se puede controlar con una única variable de tipo de dato `bool`.

b) Otra forma de implementar los cambios en el numerador y denominador es observando que en cada iteración, el numerador es el denominador de la iteración anterior más 1 y el denominador es el numerador de la iteración anterior más 1.

Ejemplo de entrada: `1000` Salida correcta: `3.1400238186006`

Ejemplo de entrada: `100000` Salida correcta: `3.14157694582286`

Finalidad: Bucles simples. Acumulación de productos. Usar un `bool` *fijado en la iteración anterior. Dificultad Media.*

55. Ampliar la funcionalidad del programa escrito para solucionar el ejercicio 49. En esta ocasión, una vez terminado un juego de adivinación podrá volverse a jugar de nuevo: el programa pedirá si queremos volver a jugar o no.

    Igual que en el ejercicio 49 considerad dos reglas de parada para caga juego: a) que haya acertado, o b) se no quiera seguir jugando a adivinar ese número (escoged cómo se quiere implementar esta opción).

    Dificultad Media.

56. Un número entero n se dice que es **desgarrable** (*torn*) si al dividirlo en dos partes `izda` y `dcha`, el cuadrado de la suma de ambas partes es igual a n.

    Por ejemplo, 88209 es desgarrable ya que $(88 + 209)^2 = 88209$. El número 81 también es desgarrable ya que $(8 + 1)^2 = 81$.

    Cread un programa que lea un entero n e indique si es o no desgarrable.

    Finalidad: Ejercitar los bucles. Dificultad Baja.

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

$$1 - \frac{1}{2} + \frac{1}{4} - \frac{1}{6} + \frac{1}{8} - \frac{1}{10} + ... - \frac{1}{2n-1} + \frac{1}{2n}$$

    para un valor n dado ($n > 0$).

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

58. Realizar un programa que presente una tabla de grados Celsius a grados Fahrenheit desde los 0 grados a los 300, con saltos de 20 grados.

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

59. Calcular mediante un programa la función **potencia** x^n con n un valor entero y x un valor real. No puede usarse la funciones de la biblioteca `cmath`.

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

60. Calcular mediante un programa la función **factorial**.

    Se dice que $n!$ es el factorial de un entero n y se define de la forma siguiente:

$$0! = 1$$
$$n! = 1 \times 2 \times 3 \times \cdots n, \ \ \forall n \geq 1$$

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

61. 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:

$$\left(\begin{array}{c} n \\ m \end{array} \right) = \frac{n!}{m!\ (n-m)!}$$

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

62. En matemáticas, la **sucesión de Fibonacci de orden 2** (a veces mal llamada *serie* de Fibonacci) es la siguiente sucesión infinita de números naturales:

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

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 (de ahí que se llame *de orden 2*). A los elementos de esta sucesión se les llama *números de Fibonacci*.

Supongamos una sucesión de orden 2. El número de Fibonacci de la posición 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 la posición n (valor introducido por el usuario).

A continuación, el programa solicitará un nuevo valor, k, y mostrará todos los números de Fibonacci $f_1, f_2, f_3, \ldots, f_k$.

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

63. El **número aúreo** 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 aúreo mediante la fórmula $a_n = \frac{f_{n+1}}{f_n}$ siendo f_n el número de Fibonacci de orden n (ver problema 62). 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 $n = 4$

Finalidad: Trabajar con bucles. Reutilizar código ya validado. Dificultad Media.

64. 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 $153 = 1^3 + 5^3 + 3^3$ y $8208 = 8^4 + 2^4 + 0^4 + 8^4$. Construir un programa que nos indique si un entero positivo es narcisista.

    Finalidad: Ejercitar los bucles. Dificultad Media.

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

$$f(x, y) = \frac{\sqrt{x}}{y^2 - 1} \quad \text{para} \quad x = -50, -48, \ldots, 50 \quad y = -40, -39, \ldots, 40$$

    es decir queremos mostrar en pantalla los valores de la función en los puntos

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

    Finalidad: Practicar con bucles anidados. Dificultad Baja.

66. Sobre la solución del ejercicio 29 de esta relación de problemas, se pide lo siguiente. Supondremos que sólo pueden introducirse intereses enteros (1, 2, 3, ...). 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 interés es 5, y un número de años es 3, hay que mostrar el capital ganado al término de cada uno de los tres años a un interés del 1 %; a continuación para un interés del 2 %, y así hasta llegar al 5 %. El programa debe mostrar:

```
Cálculos realizados al 1%:
  Dinero obtenido en el año número 1 = 2020.00
  Dinero obtenido en el año número 2 = 2040.20
  Dinero obtenido en el año número 3 = 2060.60

Cálculos realizados al 2%:
  Dinero obtenido en el año número 1 = 2040.00
  Dinero obtenido en el año número 2 = 2080.80
  Dinero obtenido en el año número 3 = 2122.42
.......
```

    Finalidad: Practicar con bucles anidados. Dificultad Baja.

67. 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 de {`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 del conjunto {`b, c, d`} y otro de {`j, k, l, m`}, es decir:

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

    Finalidad: Practicar con bucles anidados. Dificultad Baja.

68. Retome la solución del ejercicio 26 de esta misma relación de problemas y modifíquelo para leer un número entero `tope` e imprima en pantalla los divisores de todos y cada uno de los números positivos menor o iguales que `tope`.

    Finalidad: Practicar con bucles anidados. Dificultad Baja.

69. *(Examen Septiembre 2014)* ¿Cuántas veces aparece el dígito 9 en todos 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 tres enteros: `cifra` (entre 1 y 9), `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.

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

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

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

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

    es decir:

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

    Finalidad: Practicar con bucles `for` anidados. Dificultad Media.

71. 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.

    Finalidad: Practicar con bucles anidados. *Dificultad Media.*

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

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

    Finalidad: Practicar con bucles anidados. *Dificultad Media.*

73. Supongamos una serie numérica cuyo término general es:

$$a_i \; = \; a_1 r^{i-1} \; = \; a_1, a_1 r, a_1 r^2, a_1 r^3, \ldots$$

    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 usando esta fórmula. NO puede utilizarse la función `pow`.

    ¿Qué solución es preferible en términos de eficiencia?

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

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

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

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

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

75. 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 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.

76. 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, $15 = 7 + 8 = 4 + 5 + 6 = 1 + 2 + 3 + 4 + 5$.

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
```

Curiosidad: los únicos números con 0 descomposiciones son las potencias de 2.

Dificultad Media.

77. 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.

78. Ampliad el ejercicio 51 de esta misma Relación de Problemas. Ahora estamos interesados en calcular $g(x)$ y $CDF(x)$ en un intervalo de valores de la abscisa.

    Escribid un programa que pida los parámetros que definen una función gaussiana (μ y σ), los extremos del intervalo de valores de la abscisa y la diferencia (salto) entre dos valores consecutivos de x. A continuación mostrará, para cada valor de x los valores de $g(x)$ y de $CDF(x)$.

    Como indicamos en el ejercicio 51 de esta misma Relación de Problemas, para el cálculo práctico de $CDF(x)$ bastará con indicar:

    a) un valor inicial de la abscisa para el cálculo de $CDF(x)$ (tómese $\mu - 3\sigma$).

    b) un "salto" entre dos valores consecutivos de x (use una constante).

    Dificultad Media.

79. Escribir un programa en el que se presente un menú principal para que el usuario pueda elegir las siguientes opciones:

    a) Calcular factorial.

    b) Calcular potencia.

    c) Salir.

    Si el usuario elige la opción de *salir*, el programa terminará.

    Si elige *calcular potencia* se le solicitará un valor real x y un valor entero n y calculará x^n (ver ejercicio 59). Si el usuario elige *calcular factorial* se le solicita un valor entero positivo, n y a continuación calculará y mostrará $n!$ (ver ejercicio 60). Después de mostrar el valor calculado el programa mosttrará de nuevo el menú.

    Finalidad: Ejercitar los bucles anidados. *Dificultad Baja.*

80. Escribir un programa en el que se presente un menú principal para que el usuario pueda elegir entre las siguientes opciones:

    A–>Calcular adición.

    P–>Calcular producto.

    X–>Salir.

    Si el usuario elige en el menú principal:

    a) *Salir*, el programa terminará su ejecución.

    b) *Calcular adición* se presenta un menú (secundario) para que el usuario pueda elegir entre las siguientes opciones:

S–>Calcular suma.

R–>Calcular resta.

X–>Salir.

c) *Calcular producto* se presenta un menú (secundario) para que el usuario pueda elegir entre las siguientes opciones:

M–>Calcular multiplicación.

D–>Calcular división.

X–>Salir.

En las operaciones seleccionadas desde los menús secundarios el programa pedirá dos números reales, realizará la operación seleccionada, mostrará el resultado y volverá a mostrar el menú secundario seleccionado anteriormente.

En los dos menús secundarios la selección de *Salir* hace que el programa vuelva a mostrar el menú principal.

Finalidad: Ejercitar los bucles anidados. Dificultad Media.

81. Recupere la solución del ejercicio 79. Escribir un programa en el que se presente un menú principal para que el usuario pueda elegir las siguientes opciones:

   a) Introducir parámetros de la función (esperanza y desviación)

   b) Salir del programa

   Si el usuario elige la opción de salir, el programa terminará; si elige la opción de introducir los parámetros, el programa leerá los dos parámetros (esperanza y desviación).

   A continuación, el programa presentará un menú con las siguientes opciones:

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

   Si el usuario elige volver al menú anterior, el programa debe presentar el primer menú; si el usuario elige introducir los valores de abscisas, el programa le pedirá los extremos del intervalo y el incremento entre dos valores consecutivos de la abscisa, y mostrará los valores de la función gaussiana en todos los valores posibles de la abscisa (de manera similar a como se hizo en el ejercicio 78).

   Después de mostrar los valores de la función, el programa volverá al menú de introducción del rango de valores de abscisas.

   Finalidad: Ejercitar los bucles anidados. Dificultad Media.