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:

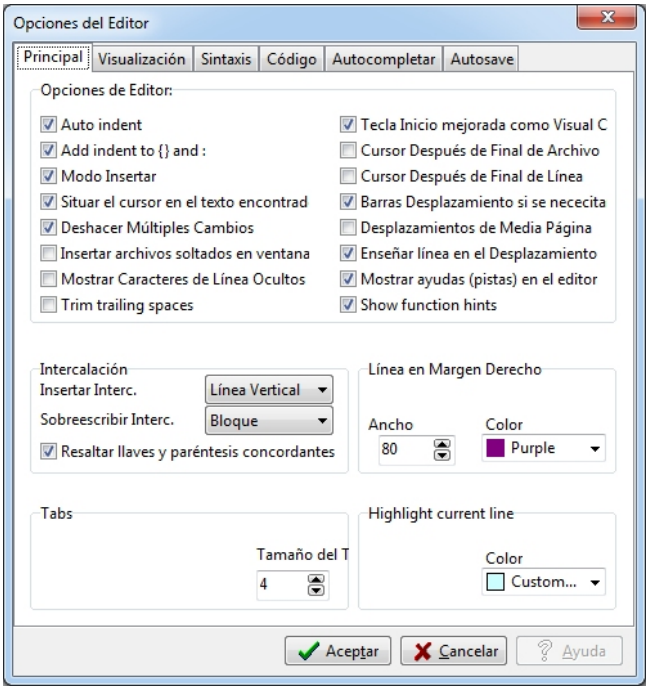
http://sourceforge.net/projects/orwelldevcpp/

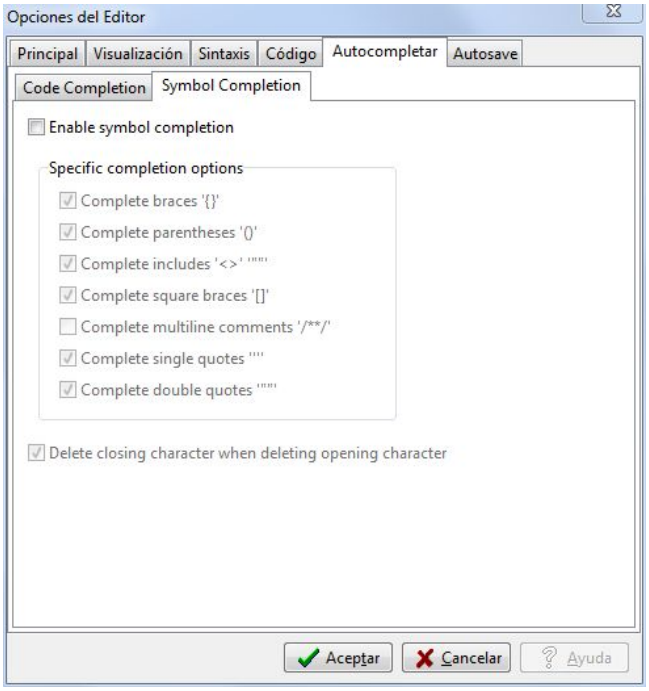
En este momento están disponibles en la misma descarga descarga las versiones 5.11 del entorno de trabajo y 4.9.2 del compilador:

- 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`) :

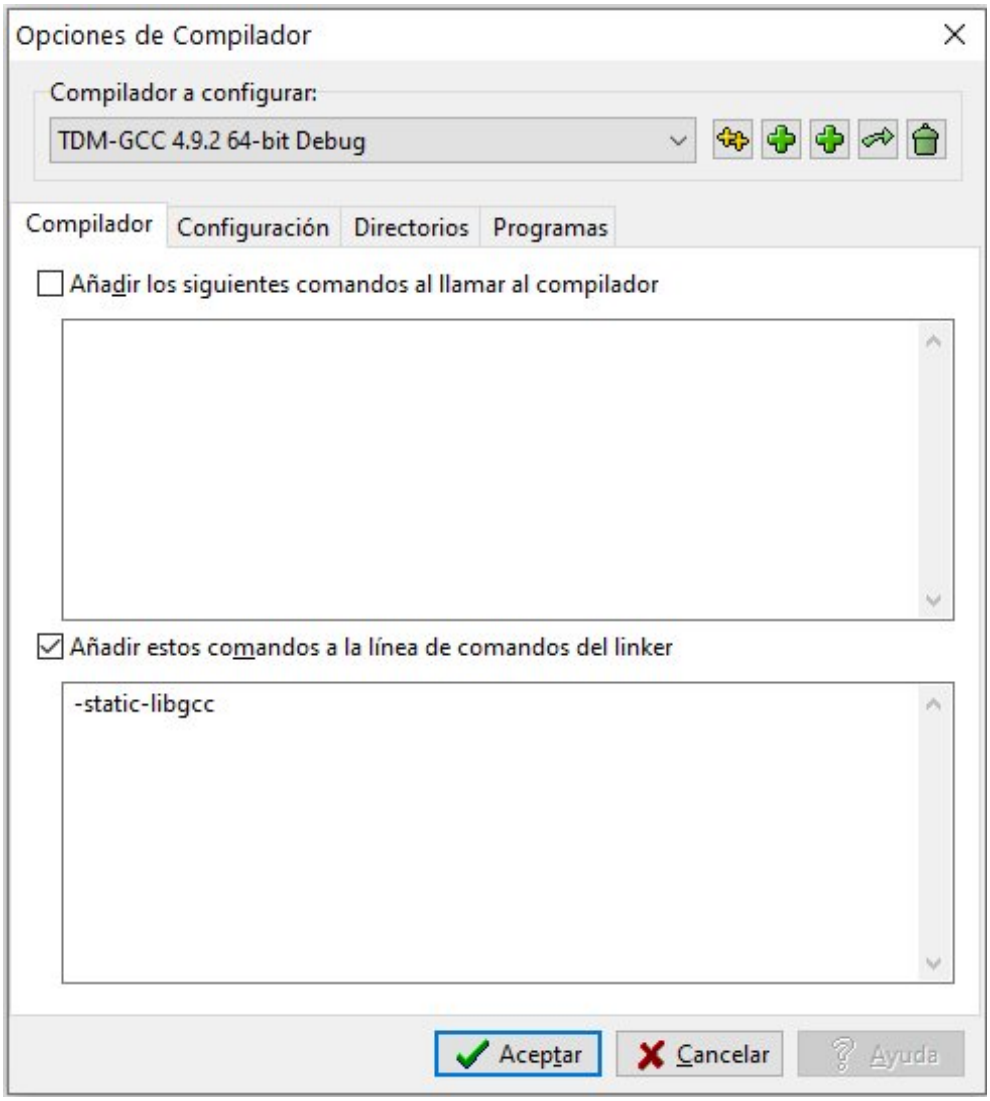
```
Herramientas -> Opciones del editor
   -> Principal
      Desmarcar Tabuladores inteligentes
      Tamaño del tabulador: 4
      Línea en Margen Derecho --> Ancho: 80  Color: Purple
   -> Autocompletar -> Symbol completion
      Desmarcar Enable Symbol completion
```





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*):

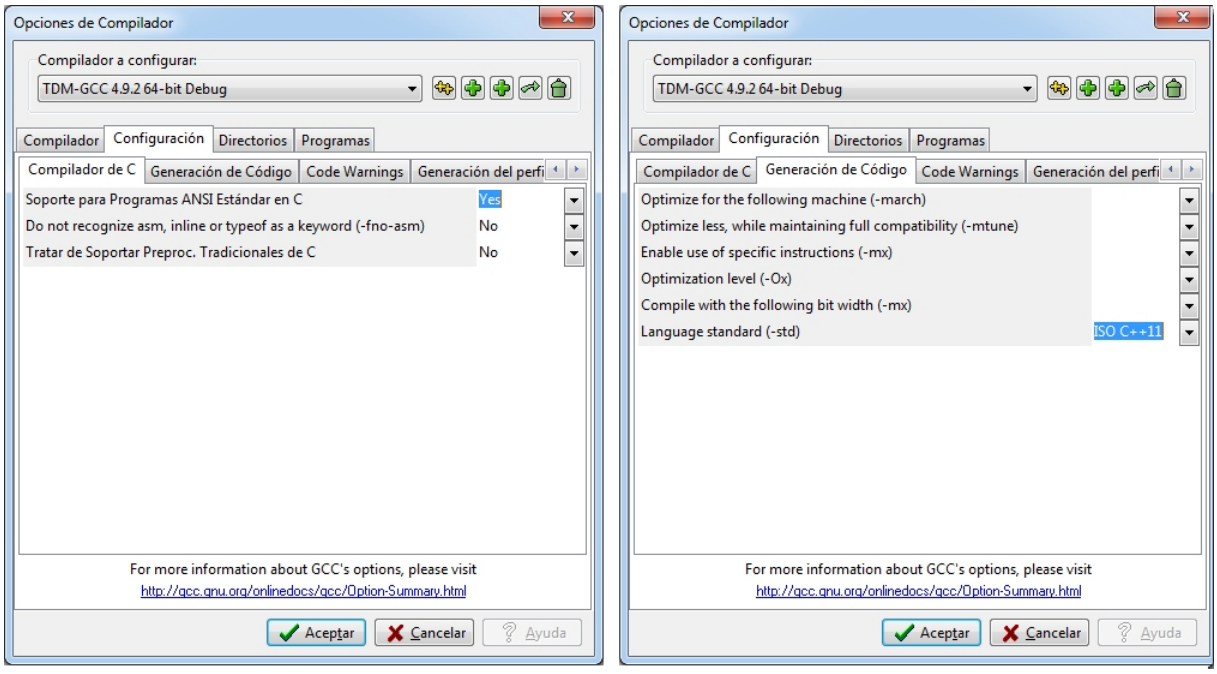


Debemos asegurarnos que se va a compilar correctamente código que sigue los estándares de programación para los lenguajes C y C++:

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

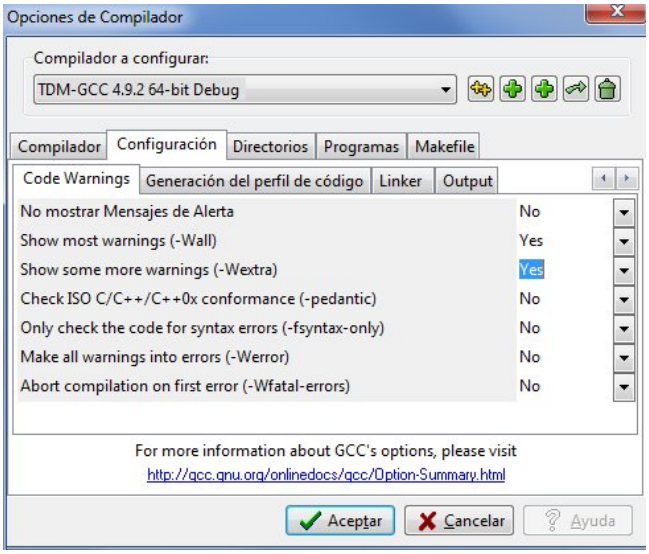
  Puede leer en Wikipedia (https://es.wikipedia.org/wiki/C%2B%2B#C++11)
  En 2011 C++11 inauguró una nueva era en la historia de C++, iniciando un ciclo trienal de lanzamiento de nuevas versiones. A C++11 le siguió C++14 y luego C++17, que es la versión actual en 2019; C++20 se encuentra próximo a estandarizarse, 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.

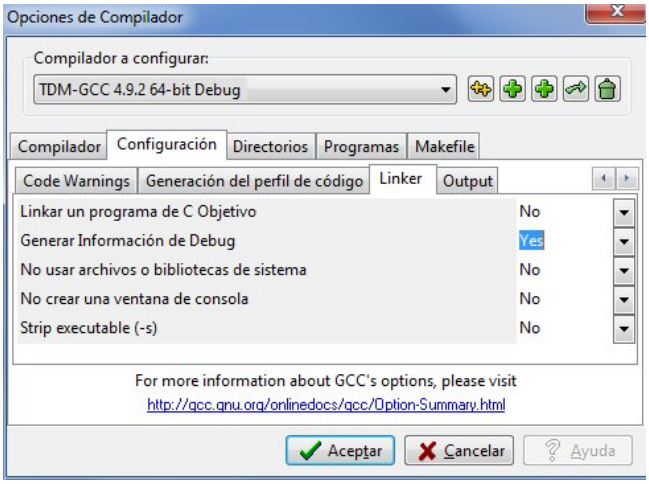
```
Herramientas -> Opciones del Compilador ->
    Configuración -> Compilador de C
        Soporte para Programas ANSI Estándar en C: Yes

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).

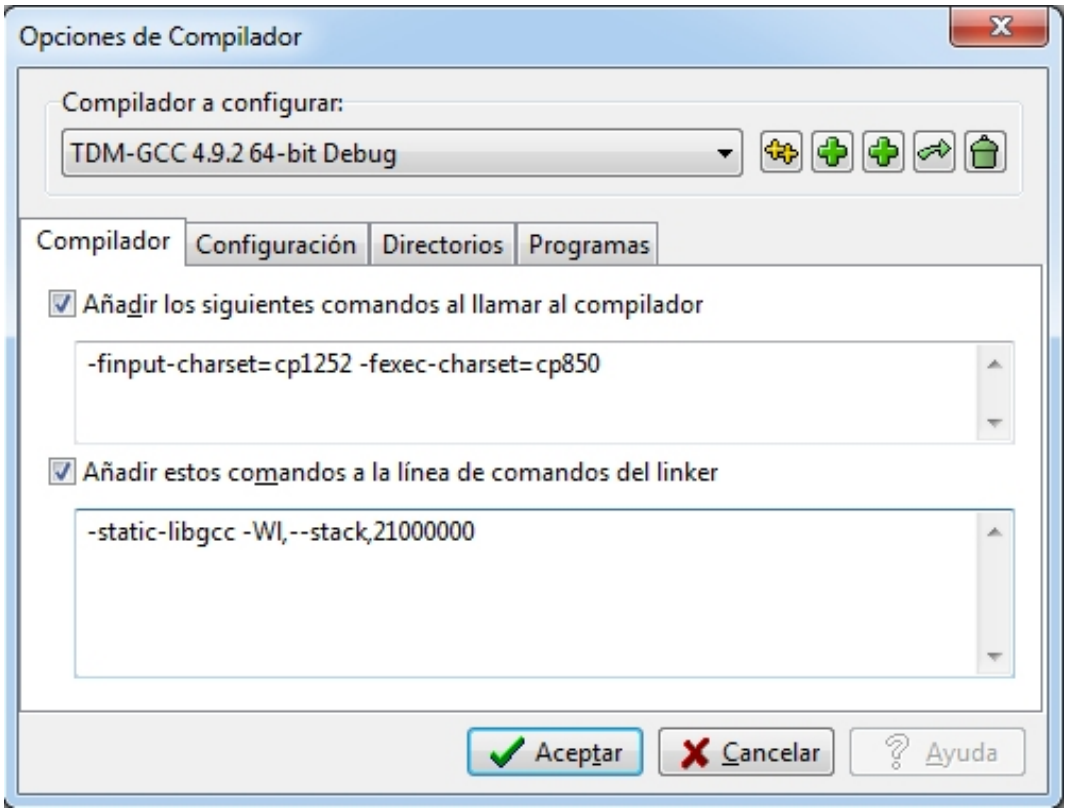


Tabla resumen de accesos directos usados en Orwell Dev C++

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 el proceso de compilación.
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:

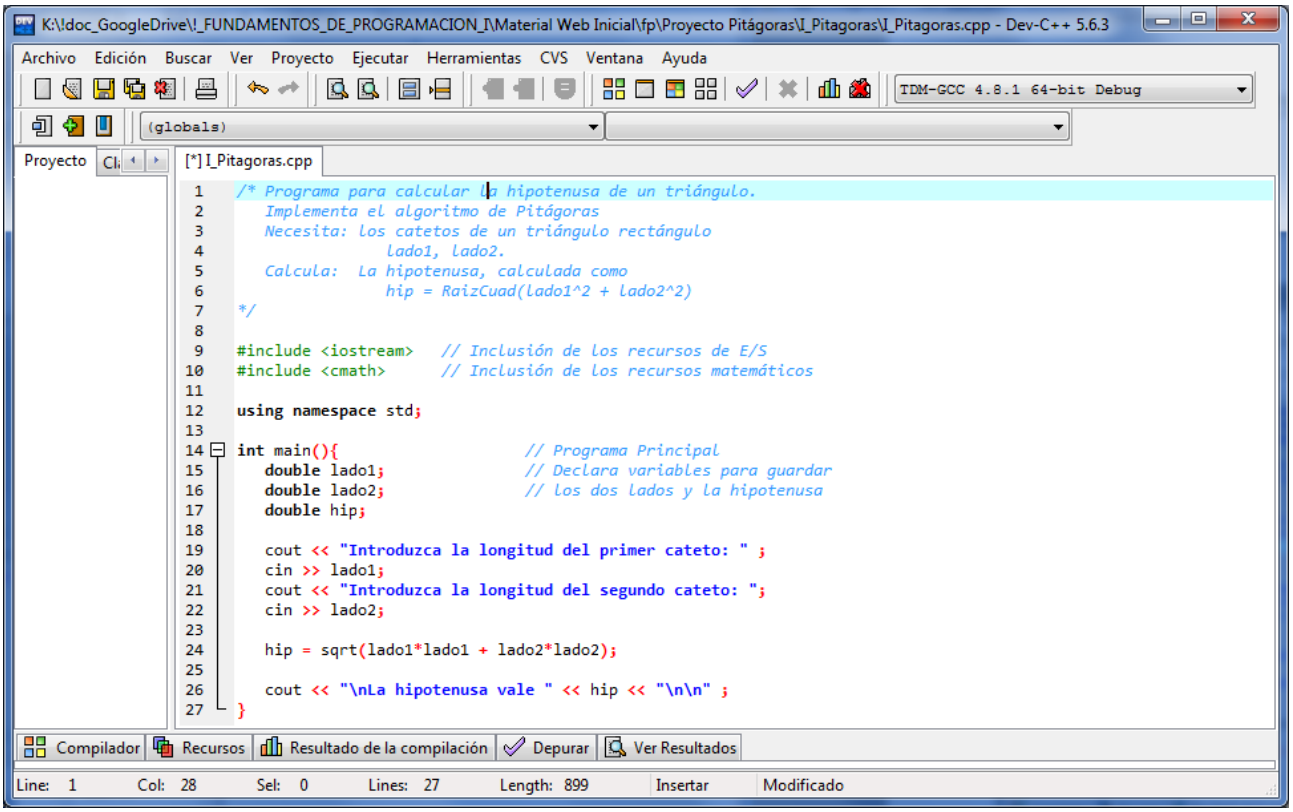


Figura 1: Programa que implementa el algoritmo de Pitágoras

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.

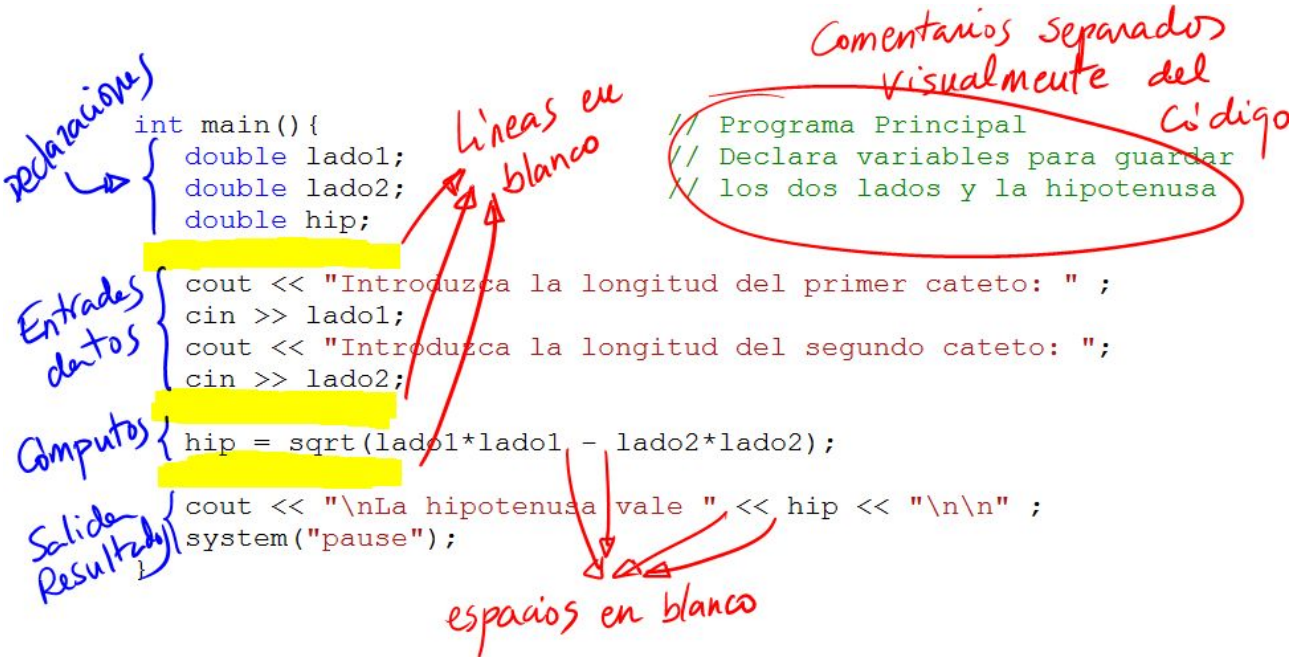


Figura 2: Escritura de código

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`).

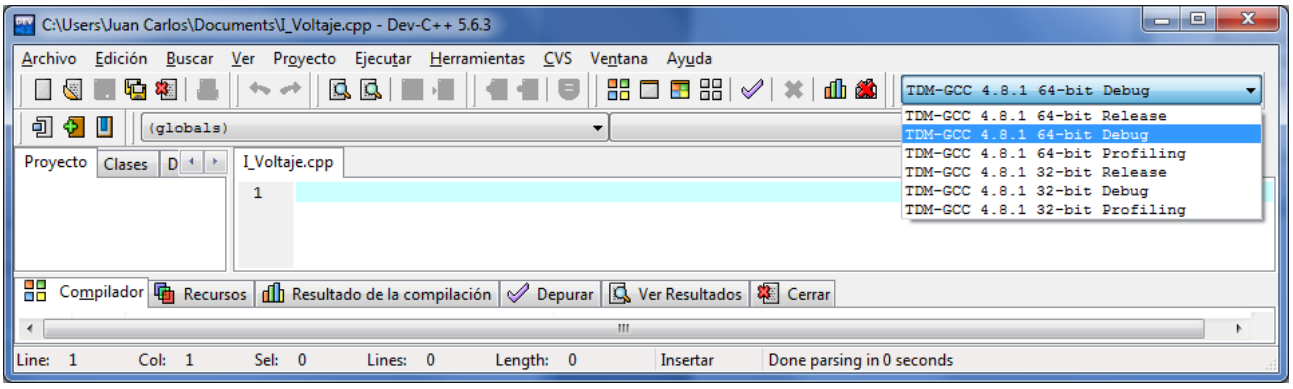


Figura 3: Creación de un programa nuevo

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.

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)

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

   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 = 1609,344 metros

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:

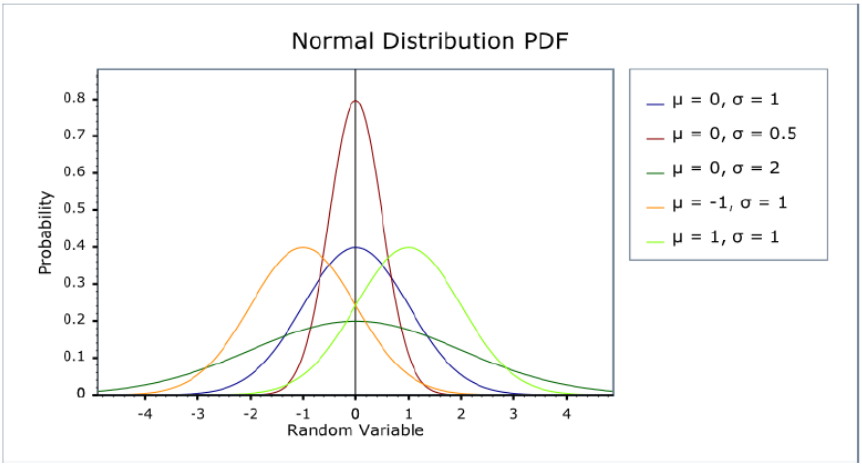
$$\text{gaussiana}(x) = \frac{1}{\sigma\sqrt{2\pi}} \; e^{\left\{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right\}}$$

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 && z > 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 $\boldsymbol{P_1} = (\boldsymbol{x_1}, \boldsymbol{y_1})$ y $\boldsymbol{P_2} = (\boldsymbol{x_2}, \boldsymbol{y_2})$ y calcule la distancia euclídea entre ellos:

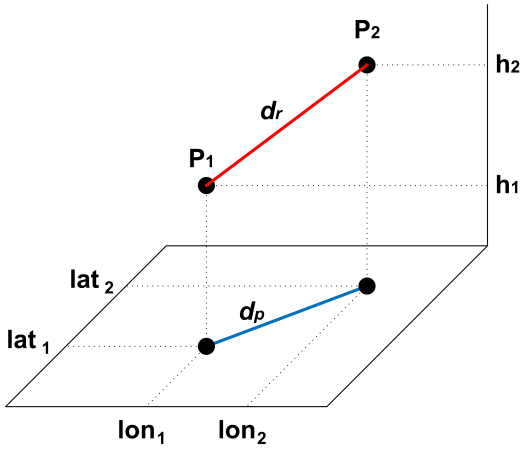
    $$\boldsymbol{d(P_1, P_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}}$$

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

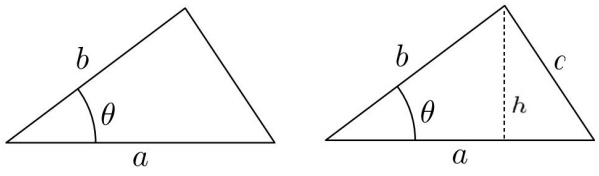
    c) $\sqrt{1+\left(\dfrac{e^x}{x^2}\right)^2}$

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 $A = \dfrac{1}{2}ab\,sin(\theta)$.

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