



Fundamentos de Programación.

Guion de Prácticas.

Curso 2022/2023

"Lo que tenemos que aprender a hacer, lo aprendemos haciéndolo".
Aristóteles



"In theory, there is no difference between theory and practice. But, in practice, there is".
Jan L. A. van de Snepscheut



"The gap between theory and practice is not as wide in theory as it is in practice".



"Theory is when you know something, but it doesn't work. Practice is when something works, but you don't know why. Programmers combine theory and practice: Nothing works and they don't know why".



Sobre el guion de prácticas

Este apartado explica las directrices generales para las clases prácticas.

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

- Las soluciones de los ejercicios deberán ser subidas a la plataforma **PRADO**. El alumno subirá un fichero zip que contendrá los ficheros con extensión `cpp` correspondientes a las soluciones de los ejercicios (no incluya los ejecutables `.exe`).

El fichero zip deberá nombrarse con el nombre y apellidos del alumno (sin espacios en blanco, acentos, etc) Por ejemplo, si su nombre es Pedro Magaña García, nómbrelo como MaganiaGarciaPedro.zip

Nombre los ficheros `cpp` según sea el ejercicio pero no use acentos ni espacios en blanco. Añádale al final el número del ejercicio. Por ejemplo: `pi_arccoseno_4.cpp`.

Incluya su nombre y apellidos como comentario al principio de cada uno de los ficheros `cpp`.

- La defensa de la sesión i se hará la semana siguiente (semana $i + 1$), durante las horas de prácticas. El profesor llamará aleatoriamente a los alumnos para que defiendan dichos ejercicios. La defensa podrá consistir en la realización en papel de alguno de los ejercicios de la sesión o explicarlo a los compañeros.

Muy importante:

La resolución de los problemas y actividades puede hacerse en grupo, pero la defensa durante las horas de prácticas es individual.

Simultáneamente a la defensa, todos los alumnos tendrán que ir realizando una serie de actividades que vienen descritas en este guion (bajo el epígrafe *Actividades a realizar en las aulas de ordenadores*). *Dichas actividades no hay que entregarlas al profesor.*

Terminada la defensa, el profesor debatirá con los alumnos las soluciones de los ejercicios.

La organización por semanas es la siguiente:

- del 12 al 16 de Septiembre: No hay clases de prácticas en las aulas de ordenadores, pero el alumno tiene que realizar en su casa una serie de actividades, descritas en la sesión 0 (ver página 9). No tiene que entregar nada.

- del 19 de Septiembre al 23 de Septiembre: Empiezan las clases de prácticas en las aulas de ordenadores. Consulte los horarios en el siguiente enlace:

<https://etsiit.ugr.es/docencia/grados>

Durante esta semana, los alumnos aprenderán a usar el entorno de programación. Además, en sus casas, los alumnos deben resolver los ejercicios indicados en la sesión 1 (ver página 19). Las soluciones a dichos ejercicios deben entregarse en PRADO antes de las 7:00h del Lunes 26 de Septiembre.

- del 26 de Septiembre al 30 de Septiembre: Durante las clases de prácticas, los alumnos defenderán los ejercicios de la sesión 1 que previamente han entregado. Además, en sus casas, los alumnos deben resolver los ejercicios indicados en la sesión 2 (ver página 21). Las soluciones a dichos ejercicios deben entregarse en PRADO antes de las 7:00h del Lunes 3 de Octubre.
- El resto de semanas se organizan de forma análoga a lo indicado en el punto anterior.

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

- **Lectura de programas resueltos**: Son ejercicios resueltos que el alumno debe leer y entender. En algunas sesiones tendrá que usar alguna de las técnicas descritas en dichos ejercicios para poder resolver alguno de los ejercicios obligatorios.
- **Obligatorios**: Todos los alumnos deben resolver y entregar estos problemas.
- **Opcionales**: Son ejercicios algo más complejos cuya entrega no es obligatoria. En cualquier caso, es muy recomendable que los alumnos intenten resolverlos para que pongan a prueba su evolución como programadores.
- **Complementarios**: No han de entregarse. Son ejercicios sencillos que pueden servir de refuerzo para aquellos alumnos que están empezando a programar.
- **Complementarios avanzados**: No han de entregarse. Son ejercicios típicos de un examen, por lo que se recomienda que el alumno intente resolverlos.

Para la realización de estas prácticas, se utilizará el entorno de programación Orwell Dev C++. En la página 4 se encuentran las instrucciones para su instalación en nuestra casa. En cualquier caso, el alumno puede instalar en su casa cualquier otro compilador como por ejemplo CodeBlocks.

Instalación de Orwell Dev C++ y de winscp en nuestra casa

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

<https://sourceforge.net/projects/orwelldevcpp/>

Cuando lo instale en su casa, debe configurarlo tal y como se indica en este apartado. En primer lugar ha de elegir en la parte superior derecha de la ventana de DevC++, el entorno de compilación que va a usar. Será algo del tipo TDM-GCC 4.9.2 64-bit Debug

```
Herramientas -> Opciones del Compilador
  Compilador
    -> Añadir los siguientes comandos al llamar al compilador
    -finput-charset=cp1252 -fexec-charset=cp850 -Wl,
    --stack,200000000
  Configuración -> Generación de código
    Language standard (-std) -> ISO C++11
  Configuración -> Code Warnings. Marcar los siguientes:
    Show most warnings
    Show some more warnings
  Configuración -> Linker
    Generar información de Debug: Yes
Herramientas -> Opciones del editor
  -> Principal
    Desmarcar Tabuladores inteligentes
    Desmarcar Permitir el uso del carácter tabulador
    Tamaño del tabulador: 3
```

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

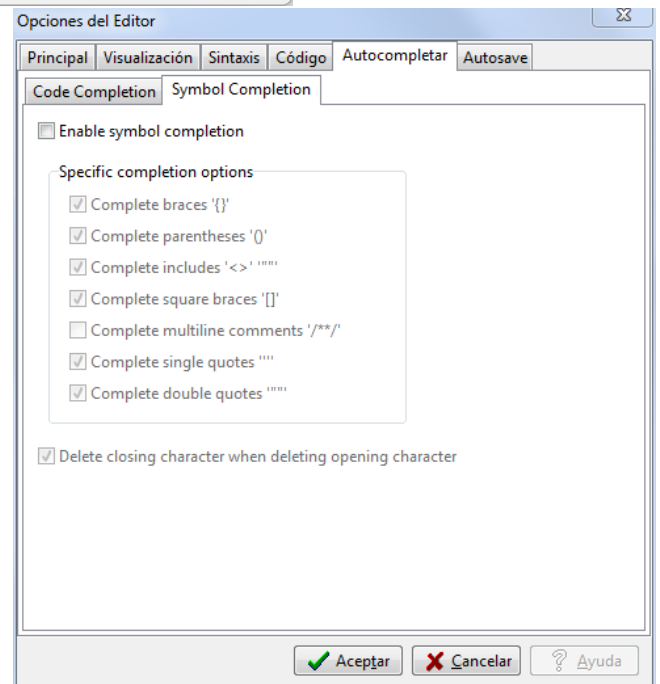
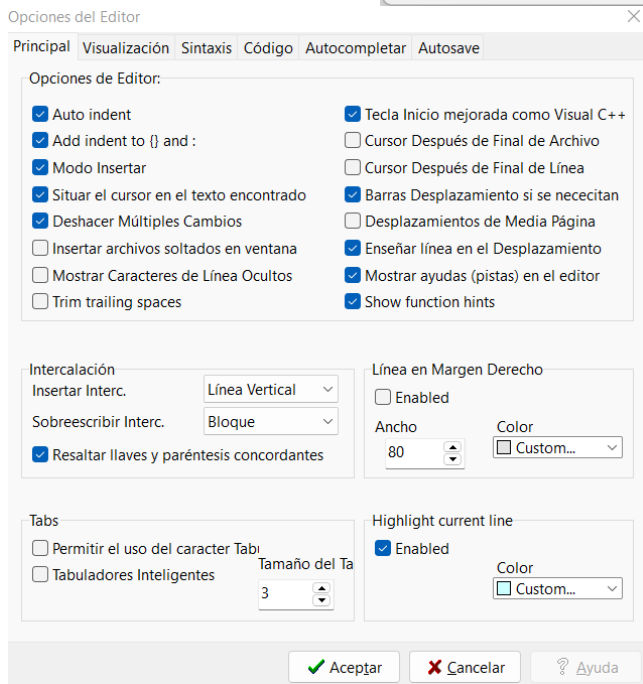
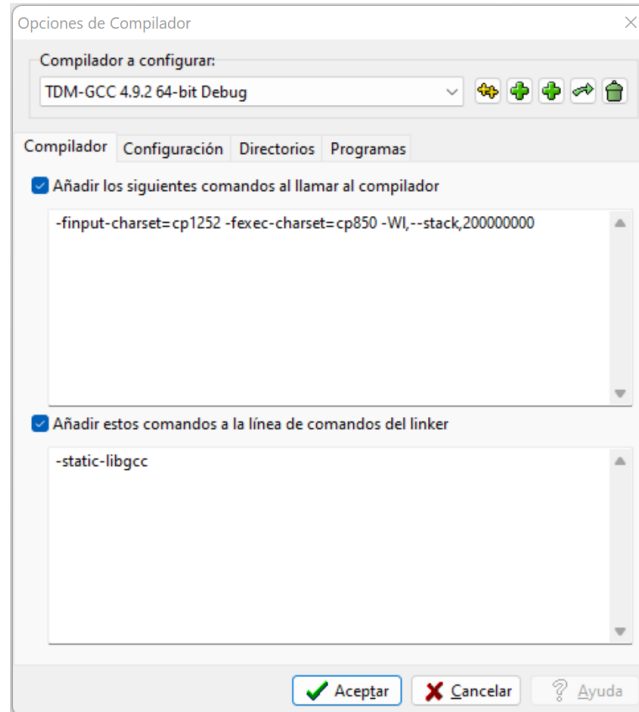
```
-finput-charset=cp1252 -fexec-charset=cp850
```

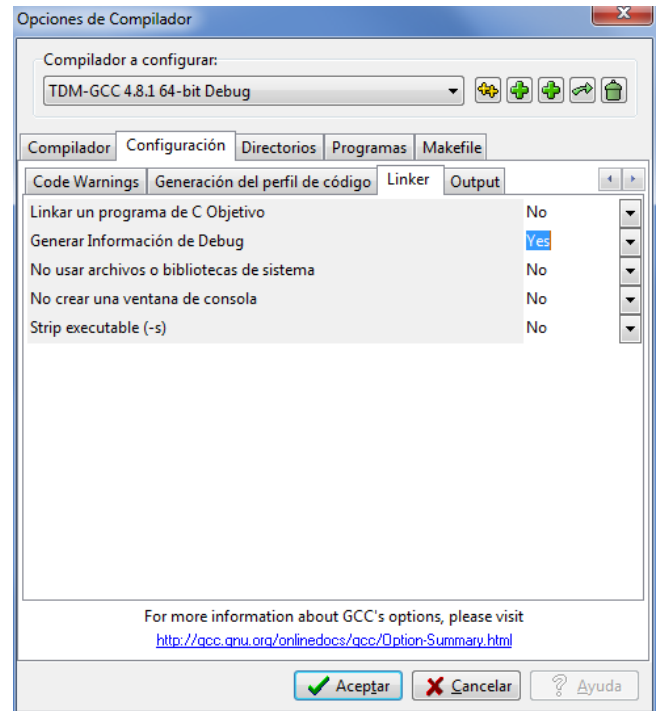
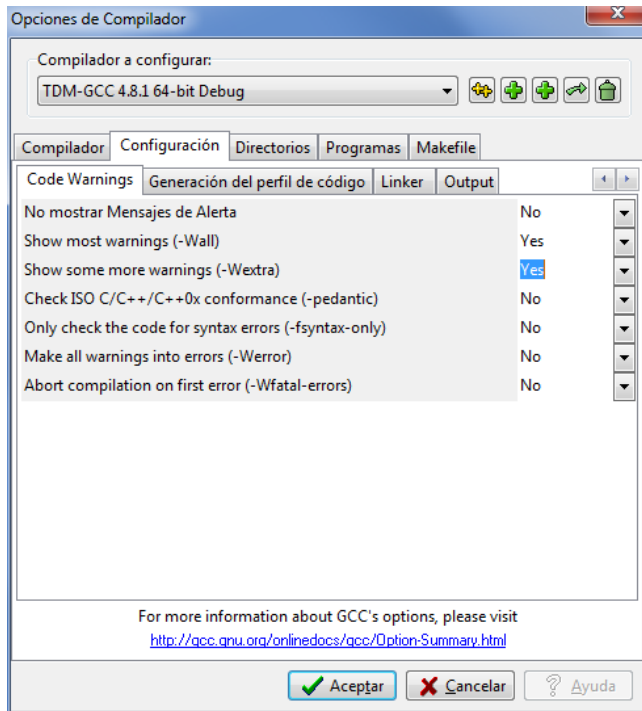
indica al compilador que el código fuente está codificado con la página de códigos 1252 (de Windows) y que el ejecutable (la consola) será compatible con la 850. De esta forma conseguimos mostrar los acentos y los caracteres especiales como la ñ en la consola que se abre al ejecutar nuestros programas.

Por otra parte, la opción

```
-Wl,--stack,200000000
```

Le indica al linker que utilice un tamaño de memoria para la pila de 200000000 bytes (ocho ceros). Esto será necesario cuando usemos vectores con tamaños muy grandes en el tema III.





Cada alumno tendrá disponible durante toda la carrera un espacio de almacenamiento en los servidores de la Escuela. Para acceder a dicho espacio, basta con seleccionar la unidad U: si está en un ordenador de la Escuela. 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, es decir, usando el mismo nombre de usuario y contraseña con el que iniciamos sesión en las clases de prácticas (ver página 9).

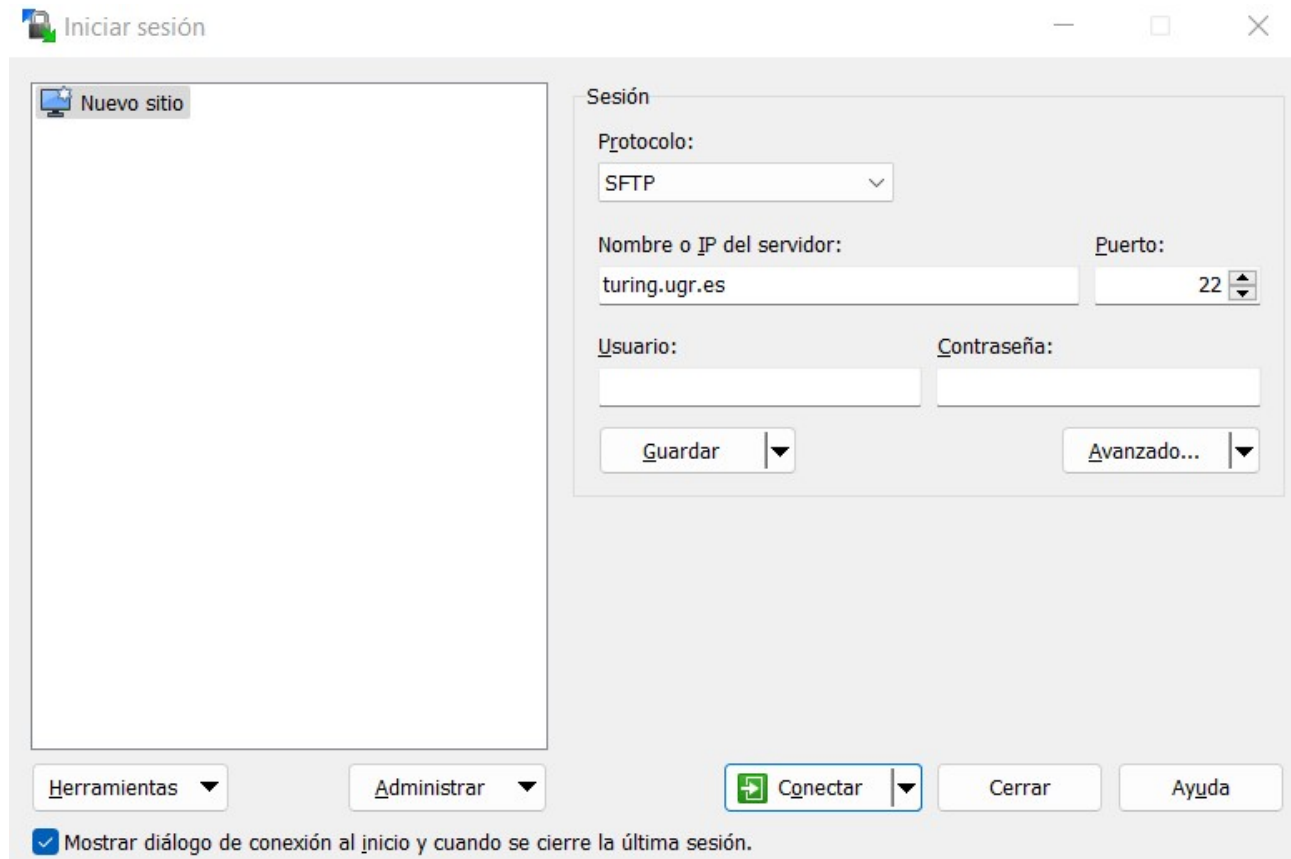


Figura 1: Conexión a la unidad U: usando winscp

En el siguiente enlace puede encontrar un video de ayuda para esta sección elaborado por el profesor Francisco José Cortijo:

<https://www.youtube.com/watch?v=CRNynkRWb7U&feature=youtu.be>

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

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

Sesión 0 (Tema I)

Esta sesión consta de:

1. *Actividades a realizar en casa.* Son las tareas que el alumno debe realizar en su casa durante la semana del 12 al 16 de Septiembre. Esta es la única sesión en la que el alumno no tendrá que realizar ninguna entrega.
2. *Actividades a realizar en las aulas de ordenadores.* Son las actividades que se realizarán con la ayuda del profesor durante la primera hora de las clases de prácticas de la semana del 19 de Septiembre al 23 de Septiembre. El alumno no tiene que haber entregado nada previamente.

► **Actividades a realizar en casa**

Durante la semana del 12 al 16 de Septiembre, el alumno debe realizar en su casa las siguientes actividades:

Actividad: Conseguir login y password.

El alumno debe conseguir una cuenta de correo electrónico de la UGR, tal y como se indica en el fichero de información general de la asignatura. Las credenciales de dicha cuenta (login y password) son las mismas que habrá que usar al arrancar los ordenadores en las aulas de prácticas.

Se recomienda que obtenga la cuenta de correo electrónico en la UGR con 48 horas de antelación al inicio de la primera sesión de prácticas (sesión 1).

Actividad: Instalación de Orwell Dev C++.

Instale en su casa el compilador Orwell Dev C++. Consulte la sección de Instalación (página 4) de este guion.

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

Estas son las actividades que se realizarán, con la ayuda del profesor, en las aulas de ordenadores durante la semana del 19 de Septiembre al 23 de Septiembre.

El Entorno de Programación. Compilación de Programas

Arranque del Sistema Operativo

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

En la casilla etiquetada como Código, introduciremos el código proporcionado por el profesor de prácticas ese mismo día. 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, tal y como se indicó en la página 7 el alumno dispone de un espacio de almacenamiento en la unidad lógica U: , cuyos contenidos permanecerán durante todo el curso académico. El alumno deberá crear el directorio U:\FP dentro de su unidad U:

En cualquier caso, es recomendable no saturar el espacio usado ya que, en caso contrario, el compilador podría no funcionar. Recuerde que las entregas de las prácticas se realizará siempre a través de PRADO . La unidad U: es una unidad de almacenamiento disponible para el alumno.

El primer programa

Copiando el código fuente

Descargue el fichero https://decsai.ugr.es/jccubero/FP/I_Pitagoras.cpp (también disponible en PRADO) y cópielo en su carpeta local (dentro de U:\FP).

Desde el Explorador de Windows, haga doble click sobre el fichero I_Pitagoras.cpp. Debe aparecer una ventana como la de la figura 2

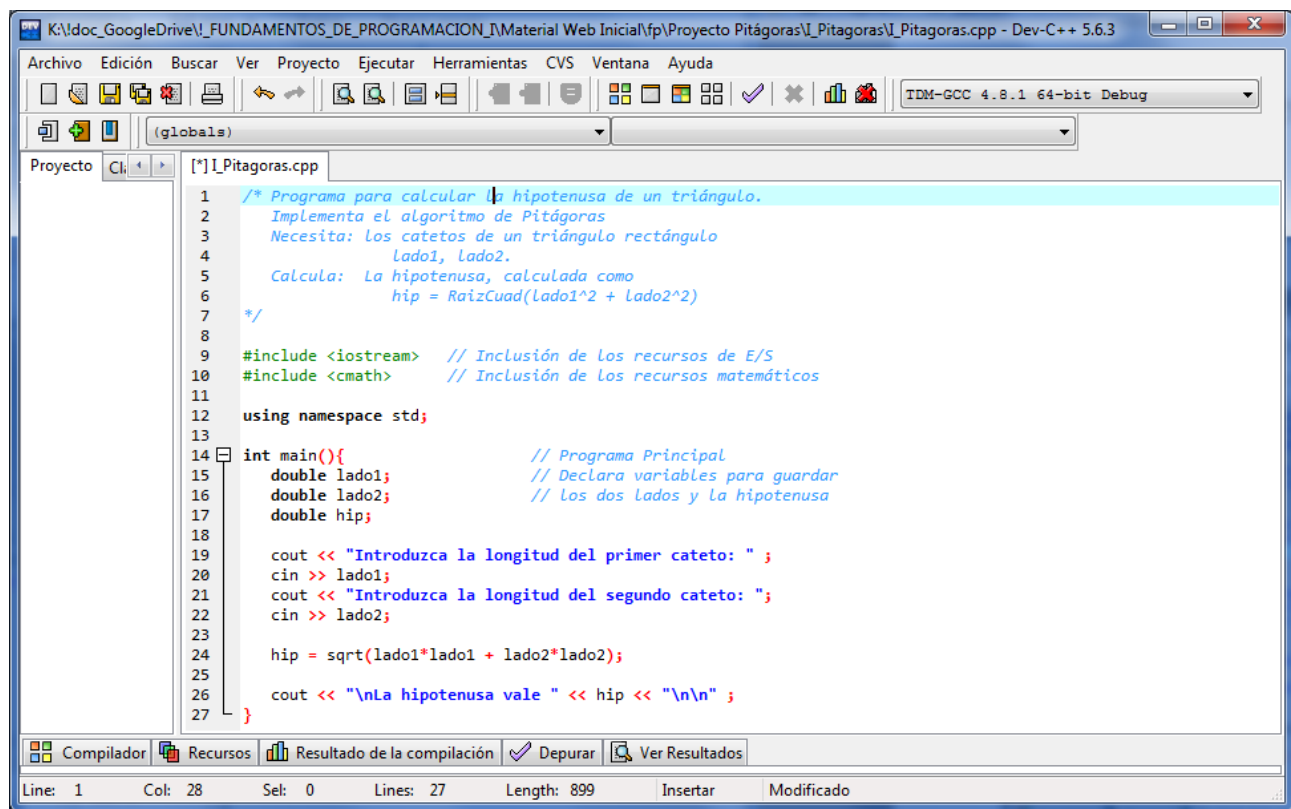


Figura 2: Programa que implementa el algoritmo de Pitágoras

Algunas consideraciones con respecto a la escritura de código (ver figura 3)

- Los programas deben incluir comentarios:
 - Comentarios que describan la idea general del algoritmo: usualmente serán comentarios de bloque (`/* . . . */`). No se debe comentar aquello que sea obvio.
 - Comentarios para explicar alguna línea concreta de código: usualmente serán comentarios de línea (`// . . .`) y no debemos abusar de ellos.

Consulte los apuntes de clase para más información.

- Es muy importante una correcta tabulación de los programas. Por ahora, incluiremos todas las sentencias del programa principal con una tabulación.
- 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. Deje también un espacio en blanco antes y después de `<<` y `>>` en las sentencias que contienen una llamada a `cout` y `cin` respectivamente.

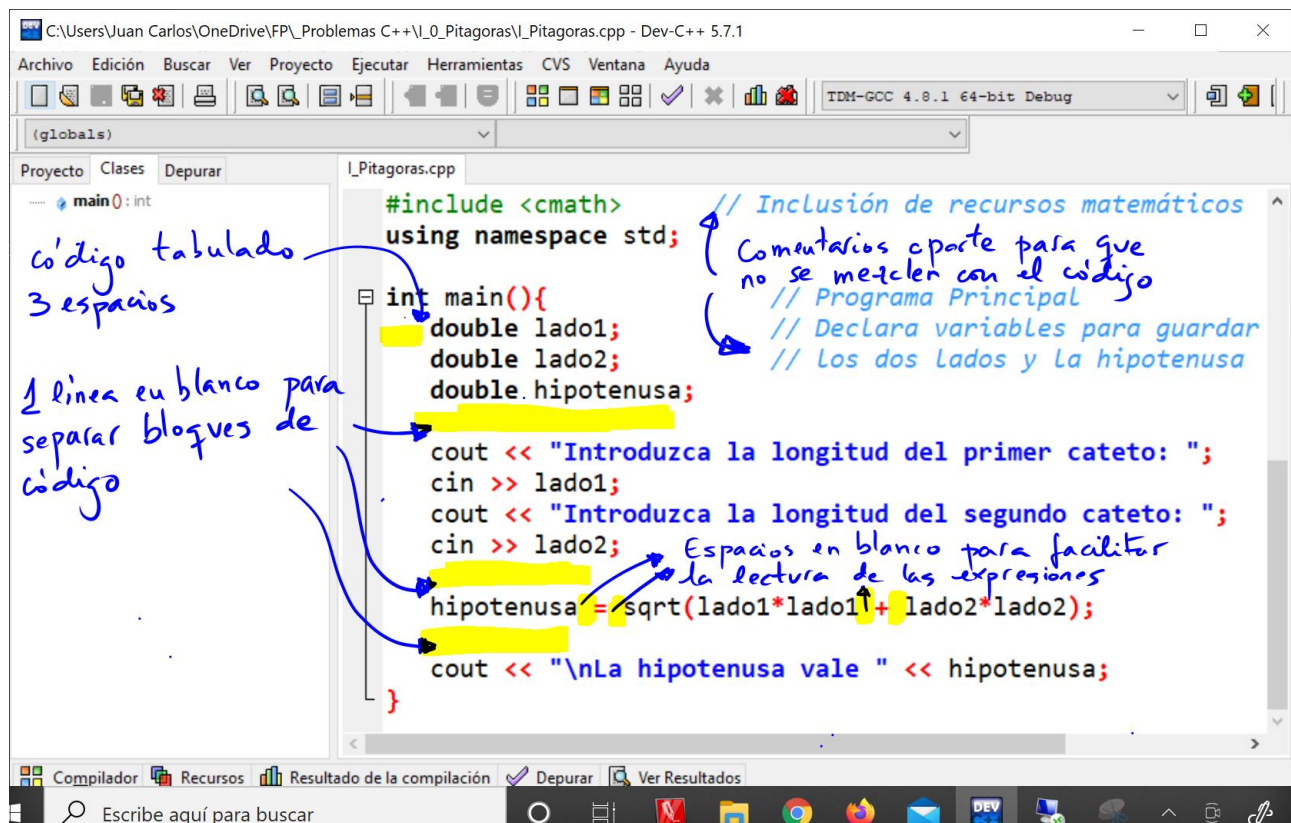



Figura 3: Escritura de código

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



Compilación

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

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

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

`Compilation succeeded`

Una vez compilado el programa, habremos obtenido el fichero `I_Pitagoras.exe`. Para ejecutarlo desde el entorno basta pulsar sobre F10. Si se quiere, ambos pasos (compilación y ejecución) pueden realizarse pulsando sobre F11. Debe aparecer una ventana de comandos del Sistema, en la que se estará ejecutando el programa. La ejecución del programa se detendrá en aquellos puntos del mismo donde se requiera la interacción del usuario para poder proseguir, es decir, en 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.

Introduzca ahora los valores pedidos en el ejemplo de Pitágoras y compruebe la respuesta del programa.

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

1. Cierre Orwell Dev C++.
2. Abra una ventana de Mi PC.

3. Sitúese en la carpeta que contiene el ejecutable.
4. Haga doble click sobre el fichero `I_Pitagoras.exe`.

Cuando lance el programa desde el sistema operativo, éste terminará cuando se ejecute la última instrucción. El proceso del sistema operativo asociado al programa terminará y se cerrará la ventana correspondiente sin dar tiempo a ver la salida de resultados. Desde el IDE no hay problema ya que éste realiza una pausa automática. Para poder realizar este pausa, o bien puede leer un dato cualquiera con `cin` o con cualquier otro recurso como `cin.get()` (se verá posteriormente)

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.

Nota:

C++11 incorpora en la biblioteca `cmath` una función específica para calcular la hipotenusa, llamada `hypot`. Si lo desea, también puede comparar los resultados de su algoritmo con el ofrecido por dicha función

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

guion de prácticas aprenderemos a corregir los errores más comunes que impiden una compilación exitosa del fichero fuente.

Cargue el fichero `I_Pitagoras.cpp`. Quítele una `'u'` a alguna aparición de `cout`. Intente 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:

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

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

1. Cambie algún punto y coma por cualquier otro símbolo
2. Cambie `double` por `dpuble`

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

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

Los errores lógicos y en tiempo de ejecución

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

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

- Cambie la sentencia

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

Ejecute 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, declare tres variables ENTERAS (tipo `int`) resultado, numerador y denominador. Asigne cero a denominador y 7 a numerador. Asigne a resultado la división de numerador entre denominador. Imprima el resultado. Al ejecutar el programa, se produce una excepción o error de ejecución al intentar dividir un entero entre cero.

Creación de un programa nuevo

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

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

Archivo->Nuevo Código Fuente (Ctrl-N)

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

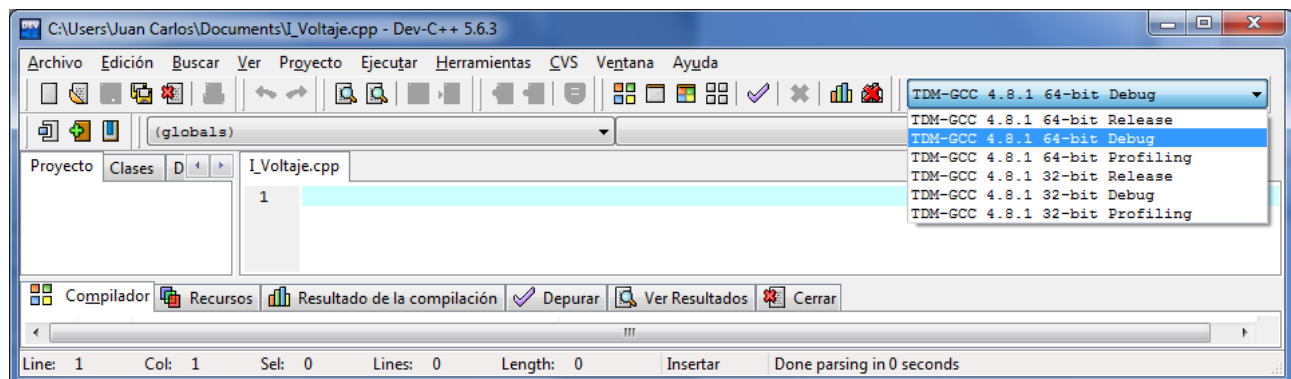


Figura 4: Creación de un programa nuevo

Importante: *No use acentos, ni espacios en blanco para nombrar el fichero.*

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

TDM-GCC ... 32 bits Debug

Normalmente, en los ordenadores de nuestras casas, tendremos que seleccionar 64 bits Debug.

Ya estamos en condiciones de resolver el problema pedido. Escribimos el código en la ventana de edición. Habrá que leer desde teclado los valores de intensidad y resistencia

y el programa imprimirá en pantalla el voltaje correspondiente. Recuerde 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.

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

Cree un fichero `cpp` por cada uno de los ejercicios. Puede ir guardándolos en su unidad `U:` para poder acceder a ellos desde su casa (recuerde lo visto en la página 7). Estos ficheros no tiene que entregarlos en este momento. Los tendrá que entregar en PRADO para defenderlos en la semana del 26 de Septiembre al 30 de Septiembre. La fecha límite para entregarlos es las 7:00h del Lunes 26 de Septiembre.

En los siguientes enlaces puede encontrar videos de ayuda para esta sección elaborados por el profesor Francisco José Cortijo:

<https://youtu.be/CRNynkRWb7U>

<https://youtu.be/ZjgLZrRZJUQ>

<https://youtu.be/fBrvYbwlhsE>

https://youtu.be/5JwzqJ_6K6E

Sesión 1 (Tema I)

► **Actividades a realizar en casa**

El alumno debe realizar estas actividades en su casa durante la semana del 19 de Septiembre al 23 de Septiembre.

Actividad: Lectura de programas resueltos.

Lea la solución de los siguientes ejercicios:

1 [Asignaciones secuenciales salario]

https://decsai.ugr.es/jccubero/FP/I_AsignacionesSecuenciales.cpp

3 [Circunferencia]

https://decsai.ugr.es/jccubero/FP/I_Circunferencia_sin_ctes.cpp

Actividad: Resolución de problemas.

Resuelva los siguientes ejercicios de la Relación de Problemas I. Recuerde que debe subir a PRADO las soluciones de estos ejercicios, siguiendo las directrices indicadas en la página 2. Debe entregarlas antes de las 7:00h del Lunes 26 de Septiembre.

Se proporciona enlaces a videos que explican el esquema de la solución de algunos ejercicios. Consulte dichos videos **DESPUÉS** de haber intentado resolver el ejercicio por sus propios medios.

- *Obligatorios:*

4 [Cálculo de π a partir del arco-seno]

https://decsai.ugr.es/jccubero/FP/I_Pi.mp4

5 [Variación porcentual]

https://decsai.ugr.es/jccubero/FP/I_VariacPorcentual.mp4

6 [Subir sueldo]

https://decsai.ugr.es/jccubero/FP/I_SubirSueldo.mp4

9 [Conversión de grados a radianes]

https://decsai.ugr.es/jccubero/FP/I_Grados_to_Radianes.mp4

- *Opcionales:*

10 [Decimal redondeado]

https://decsai.ugr.es/jccubero/FP/I_Round.mp4

- *Complementarios:*

Los siguientes ejercicios no tiene que entregarlos ni se defenderán. Si está empezando a programar le podrán ayudar a afianzar conceptos básicos.

7 [Interés bancario]

https://decsai.ugr.es/jccubero/FP/I_InterBancario.mp4

8 [Distancia Euclídea]

https://decsai.ugr.es/jccubero/FP/I_DistanciaEuclidea.mp4

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

Estas actividades se realizan en el aula de prácticas durante la semana del 26 de Septiembre al 30 de Septiembre.

El profesor irá corrigiendo individualmente (a algunos alumnos elegidos aleatoriamente) los ejercicios encargados para esta semana y que el alumno ya ha tenido que entregar en PRADO . Mientras tanto, los alumnos restantes deben intentar resolver los ejercicios encargados para la próxima semana (los indicados en las *Actividades a realizar en casa* de esta sesión)

Sesión 2 (Tema I)

► **Actividades a realizar en casa**

El alumno debe realizar estas actividades en su casa durante la semana del 26 de Septiembre al 30 de Septiembre.

Actividad: Lectura de programas resueltos.

Lea la solución de los siguientes ejercicios:

14 [Uso de constantes] (Sólo la Circunferencia)

https://decsai.ugr.es/jccubero/FP/I_Circunferencia.cpp

16 [Sistema métrico]

https://decsai.ugr.es/jccubero/FP/I_ConversionSistemaMetrico.cpp

21 [Índice de mayúscula]

https://decsai.ugr.es/jccubero/FP/I_IndiceMayuscula.cpp

Actividades de Ampliación (no son obligatorias y no forman parte de la evaluación)



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

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

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

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

Actividad: Resolución de problemas.

Recuerde que debe subir a **PRADO** las soluciones de estos ejercicios siguiendo las directrices indicadas en la página 2. Debe entregarlas antes de las 7:00h del Lunes 3 de Octubre.

Se proporciona enlaces a videos que explican el esquema de la solución de algunos ejercicios. Consulte dichos videos **DESPUÉS** de haber intentado resolver el ejercicio por sus propios medios.

- **Obligatorios:**

- 11 [Tarifa aérea según km]

- https://decsai.ugr.es/jccubero/FP/I_Tarifa_km.mp4

- 12 [Tarifa aérea con descuento]

- https://decsai.ugr.es/jccubero/FP/I_TarifaDescuento.mp4

- 13 [Segundos entre dos instantes]

- https://decsai.ugr.es/jccubero/FP/I_Instantes.mp4

- 15 [Intercambiar variables]

- https://decsai.ugr.es/jccubero/FP/I_Intercambiar.mp4

- 25 [Pasar de mayúscula a minúscula]

- https://decsai.ugr.es/jccubero/FP/I_ToMinuscula.mp4

- **Opcionales:**

- 18 [Intercambiar tres variables]

- https://decsai.ugr.es/jccubero/FP/I_Intercambiar3.mp4

- 29 [Precisión y desbordamiento]

- https://decsai.ugr.es/jccubero/FP/I_PrecisionDesbordamiento.mp4

- **Complementarios:**

- Los siguientes ejercicios no tiene que entregarlos ni se defenderán. Si está empezando a programar le podrán ayudar a afianzar conceptos básicos.*

- 14 [Uso de constantes]

- 17 [Gaussiana]

- https://decsai.ugr.es/jccubero/FP/I_Gaussiana.mp4

- 23 [Trunca decimales]

- https://decsai.ugr.es/jccubero/FP/I_TruncaDecimales.mp4

- 24 [Lectura de un intervalo]

- 28 [Elección tipo de dato]

- https://decsai.ugr.es/jccubero/FP/I_EleccionTipoDato.mp4

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

Estas actividades se realizan en el aula de prácticas durante la semana del 3 al 7 de Octubre.

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

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

Como base para esta práctica vamos a emplear la solución preliminar del ejemplo de la ecuación de segundo grado visto en clase.

https://decsai.ugr.es/jccubero/FP/II_Ec2Grado_vs0.cpp

En primer lugar, copiamos en nuestra carpeta local el anterior fichero. Fuerce los siguientes errores en tiempo de compilación, para ver los mensajes de error ofrecidos por el compilador:

- Suprima los paréntesis de la expresión lógica del condicional (los que hay después del `if`).
- Quite la llave abierta `{` del `if`.
- Quite la llave cerrada `}` del `if`.
- Quite la llave abierta `{` del `else`.
- Quite la llave abierta `{` y la llave cerrada `}` del `else`. ¿Qué ocurre?

El buffer de entrada

Para comprobar el correcto funcionamiento de nuestros programas, tendremos que ejecutarlos en repetidas ocasiones usando distintos valores de entrada. Este proceso lo repetiremos hasta que no detectemos fallos. Vamos a ver cómo simplificar este proceso.

Cada vez que se ejecuta `cin`, se lee un dato desde el periférico por defecto. Si en la ejecución de una sentencia `cin` introducimos varios valores separados por espacios en blanco, todos ellos se almacenan dentro de una zona de memoria intermedia denominada buffer. Este buffer canaliza el flujo de datos entre el programa y el dispositivo de entrada de

datos, de forma que las sentencias `cin` posteriores leerán los datos que hay en el buffer. Para más detalle, consulte el final del Tema 1, disponible en [PRADO](#).

Esto nos permite copiar todos los datos de entrada que mi programa necesita (separados por espacios en blanco) y pegarlos en la ejecución de la primera sentencia `cin`. Las ejecuciones posteriores de las sentencias `cin` irán leyendo los datos que hay almacenados en el buffer. Cuando el buffer esté vacío, automáticamente se solicitarán más datos al dispositivo de entrada de datos.

Para comprobarlo, copie localmente el fichero https://decsai.ugr.es/jccubero/FP/II_cin.cpp. El programa simplemente lee un entero y dos caracteres y los imprime en pantalla. Cree un fichero de texto con un entero y dos caracteres. Separe estos tres datos con varios espacios en blanco. Seleccione con el ratón los tres y cópielos al portapapeles (Click derecho-Copiar). Ejecute el programa y cuando aparezca la consola del sistema haga click derecho sobre la ventana y seleccione **Editar-Pegar**. También puede usar la típica combinación de teclas `Ctrl-C` para copiar y `Ctrl-V` para pegar.

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

Sesión 3 (Temas I y II)

► **Actividades a realizar en casa**

Actividad: Lectura de programas resueltos.

Lea la solución de los siguientes ejercicios:

De la Relación de Problemas I:

19 [Media y desviación]

https://decsai.ugr.es/jccubero/FP/I_Media.cpp

De la Relación de Problemas II:

1 [Mismo signo -multiplicando-]

https://decsai.ugr.es/jccubero/FP/II_MismoSignoMultiplicando.cpp

9 [Convertir mayúscula en minúscula]

https://decsai.ugr.es/jccubero/FP/II_Mayuscula.cpp

10 [Pasar de mayúscula a minúscula y viceversa]

https://decsai.ugr.es/jccubero/FP/II_MayusculaMinusculaViceversa.cpp

Actividad: Resolución de problemas.

Se proporciona enlaces a videos que explican el esquema de la solución de algunos ejercicios. Consulte dichos videos **DESPUÉS** de haber intentado resolver el ejercicio por sus propios medios.

- *Obligatorios:* (de la relación de Problemas I)

26 [Pasar de carácter a entero]

https://decsai.ugr.es/jccubero/FP/I_Caracter_to_Entero.mp4

27 [Expresiones lógicas]

https://decsai.ugr.es/jccubero/FP/I_ExpresionesLogicas.mp4

Obligatorios: (de la relación de Problemas II)

2 [Se dividen]

https://decsai.ugr.es/jccubero/FP/II_SeDividen.mp4

4 [Tarifa aérea]

https://decsai.ugr.es/jccubero/FP/II_TarifaAerea.mp4

6 [Operadores lógicos]

https://decsai.ugr.es/jccubero/FP/II_OperadoresLogicos.mp4

7 [Tarifa aérea con descuentos]

[https://decsai.ugr.es/jccubero/FP/II_TarifaAereaDescuentos.m
p4](https://decsai.ugr.es/jccubero/FP/II_TarifaAereaDescuentos.mp4)

- *Opcionales:*

11 [Mayoría absoluta]

https://decsai.ugr.es/jccubero/FP/II_MayoriaAbsoluta.mp4

- *Complementarios:*

Los siguientes ejercicios no tiene que entregarlos ni se defenderán. Si está empezando a programar le podrán ayudar a afianzar conceptos básicos.

De la relación de Problemas I:

20 [Aproximación del valor de π]

22 [Descuento tarifa aérea mezclando tipos]

De la relación de Problemas II:

3 [Año bisiesto]

https://decsai.ugr.es/jccubero/FP/II_Bisiesto.mp4

5 [Comparación de dos instantes (restando segundos)]

Actividades de Ampliación (no son obligatorias y no forman parte de la evaluación)



Hojea la página

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

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

► Actividades a realizar en las aulas de ordenadores

Depuración

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



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

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



Para poder realizar tareas de depuración en Dev C++ debemos asegurarnos que estamos usando un perfil del compilador con las opciones de depuración habilitadas. Si cuando configuramos el compilador seleccionamos Herramientas | Opciones del Compilador | Compilador a configurar: Debug nuestro entorno estará preparado para depurar programas.

Si no fuera así, al intentar depurar el programa, Dev C++ nos mostrará la ventana de la figura 5.

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 pasos que se detallan a continuación. Vamos a usar como ejemplo base el de la ecuación de segundo grado:

https://decsai.ugr.es/jccubero/FP/II_Ec2Grado_vs0.cpp

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:

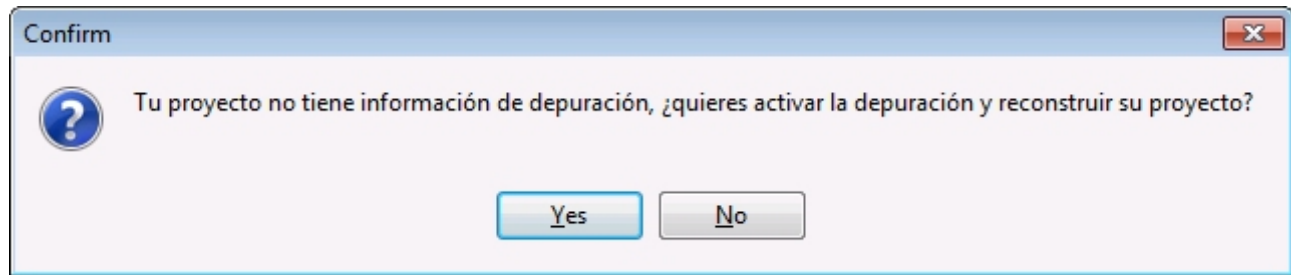



Figura 5: Ventana emergente que aparece cuando la configuración actual del compilador no permite tareas de depuración

- 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.

Coloque ahora un punto de ruptura sobre la línea que contiene la sentencia condicional **if** (figura 6).

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 7)

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

Al iniciarse la depuración se ejecutan todas las sentencias hasta alcanzar el primer punto de ruptura. Llegado a este punto, la ejecución se interrumpe (queda “en espera”) y se muestra en azul (figura 8) la línea que se va a ejecutar a continuación (en este caso, la que contiene el punto de interrupción).

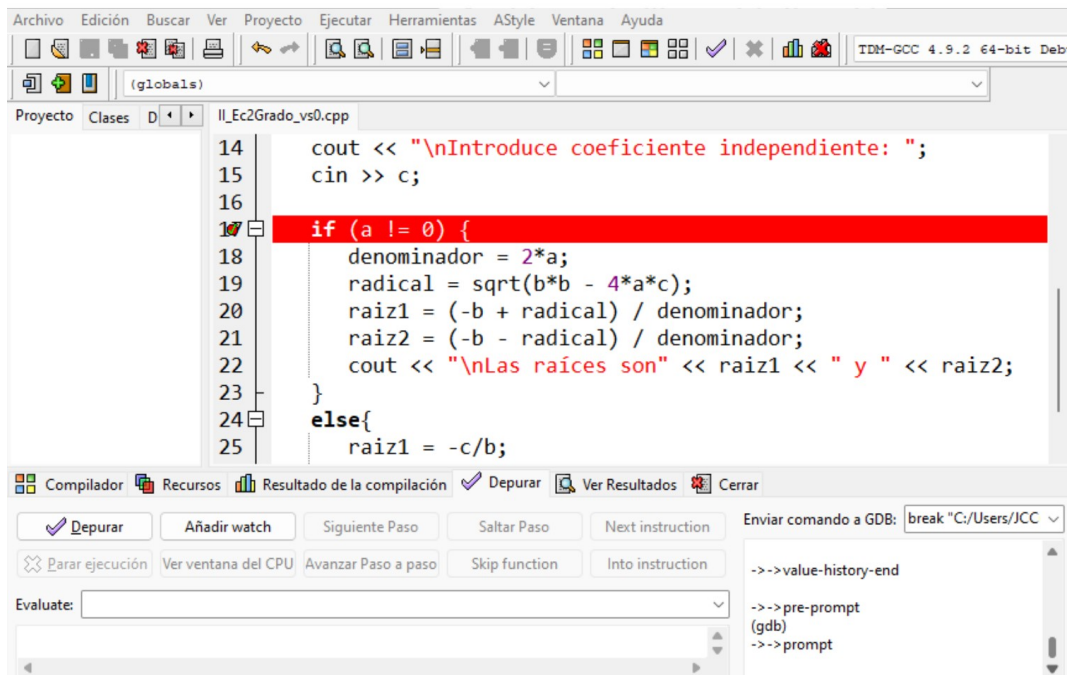


Figura 6: Se ha activado un punto de ruptura

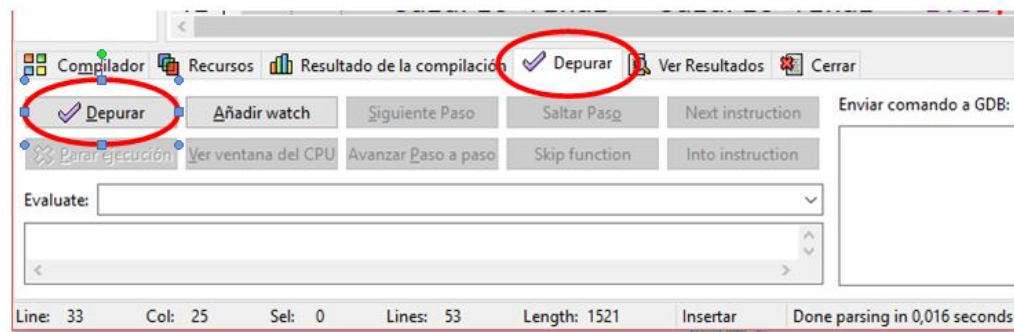


Figura 7: Inicio del proceso de depuración

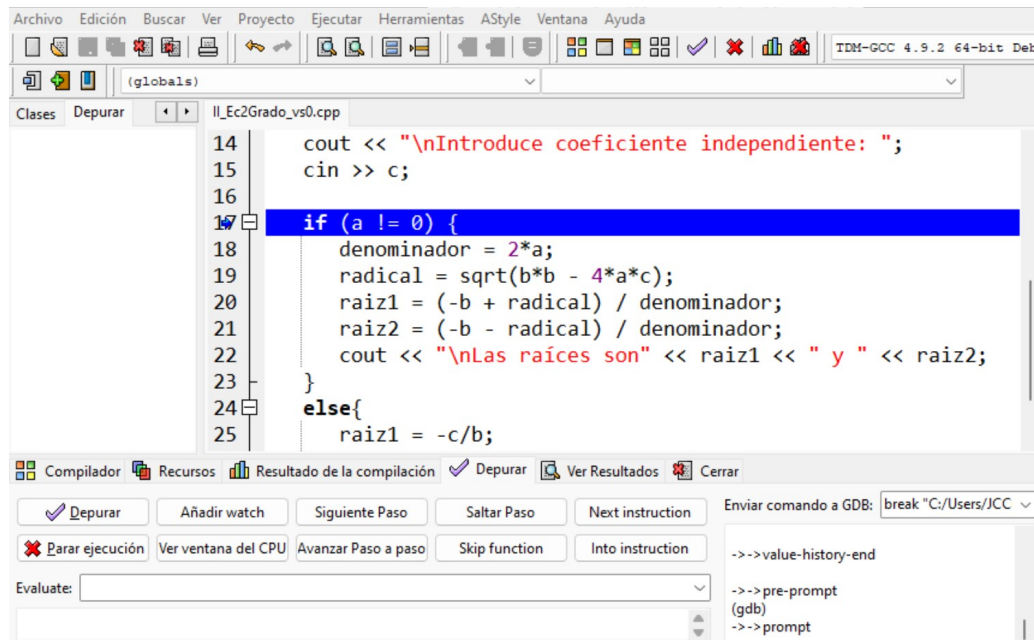


Figura 8: Inicio del proceso de depuración

Ahora podemos escoger entre varias alternativas, todas ellas accesibles en la zona inferior (pestaña Depurar) pulsando el botón correspondiente (ver figura 8):

- **Parar ejecución**: Detener la depuración (y ejecución) del programa.
- **Siguiente Paso (F7)**: Ejecuta la siguiente instrucción. Si se trata de una llamada a una función, la ejecuta y continúa con la siguiente instrucción, sin entrar a ejecutar las instrucciones internas de la función. Las funciones se verán dentro de dos semanas.
- **Avanzar Paso a paso (F8)**: Ejecuta la siguiente instrucción. Si se trata de una llamada a una función, entra en la función y ejecuta la primera instrucción de la función, continuando la depuración dentro de la función.
- **Saltar Paso**: Ejecuta todas las instrucciones hasta encontrar un nuevo punto de ruptura, o llegar al final del programa.
- **Importante**. No debe abortarse la ejecución del programa, cerrando la ventana de la consola (a través de **Ctrl-C** o haciendo click sobre la aspa de la ventana). Esta forma de parar la ejecución es *brusca* y el compilador podría no liberar adecuadamente algunos recursos utilizados del sistema operativo.

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. 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.

Podemos mantener variables permanentemente monitorizadas. Aparecerán en el Explorador de Proyectos (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. Al seleccionar OK aparece la información de esa variable en el Explorador de Proyectos (figura 9)

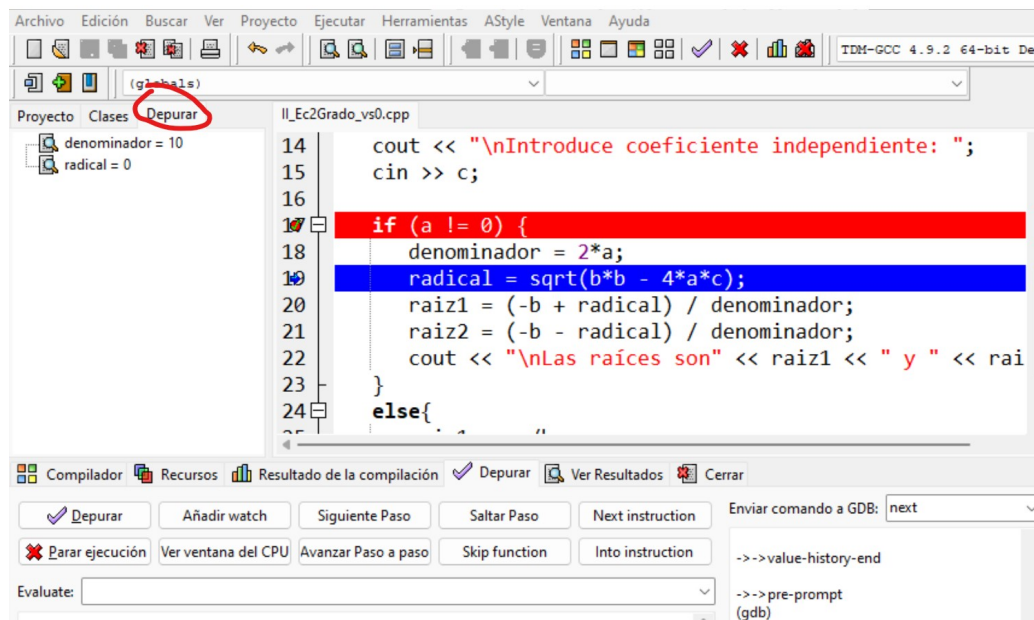


Figura 9: Añadiendo una variable para su inspección permanente

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 (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 (pestaña Depurar), son **Quitar watch** para eliminar una variable y **Clear All** para eliminarlas todas,

Practique lo visto anteriormente a un error que se dará con frecuencia: usar una variable no asignada previamente. Para ello, elimine la sentencia en la que se le asigna un valor a la variable `denominador` y ejecute el programa. Los valores de salida serán indeterminados ya que la variable `denominador` no se le ha asignado ningún valor. Añada un **watch** de dicha variable y ejecute paso a paso el programa para comprobar que, efectivamente, el valor de `denominador` es indeterminado.

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".



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

Sesión 4 (Tema II)

► **Actividades a realizar en casa**

Actividad: Seminario.

Visione el siguiente video (El tipo de dato enumerado):

https://decsai.ugr.es/jccubero/FP/II_TipoEnumerado.mp4

Lea y entienda la solución al ejercicio de la ecuación de segundo grado con enumerados:

https://decsai.ugr.es/jccubero/FP/II_Ec2gradoEnum.cpp

Actividad: Lectura de programas resueltos.

Lea la solución de los siguientes ejercicios:

17 [Mismo signo con condicionales]

https://decsai.ugr.es/jccubero/FP/II_MismoSigno.cpp

Actividad: Resolución de problemas.

Se proporciona enlaces a videos que explican el esquema de la solución de algunos ejercicios. Consulte dichos videos **DESPUÉS** de haber intentado resolver el ejercicio por sus propios medios.

- **Obligatorios:**

Ejercicios sobre condicionales:

8 [Tabulación]

12 [Velocidad imputada]

https://decsai.ugr.es/jccubero/FP/II_VelocidadImputada.mp4

15 [Intervalo]

https://decsai.ugr.es/jccubero/FP/II_Intervalo.mp4

18 [Mayúscula a minúscula y viceversa usando un enumerado]

https://decsai.ugr.es/jccubero/FP/II_MayMinEnum.mp4

Ejercicios sobre bucles:

21 [Valores de la Gaussiana]

https://decsai.ugr.es/jccubero/FP/II_Gaussiana.mp4

- **Opcionales:**

14 [Comparación de dos instantes]

https://decsai.ugr.es/jccubero/FP/II_CompararInstantes.mp4

- **Complementarios:**

Los siguientes ejercicios no tiene que entregarlos ni se defenderán. Si está empezando a programar le podrán ayudar a afianzar conceptos básicos.

Ejercicios sobre condicionales:

13 [Multa autovía]

Ejercicios sobre bucles:

19 [Divisores de un entero]

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

Bájese el siguiente programa:

https://decsai.ugr.es/jccubero/FP/II_DivisoresMAL.cpp

Este programa tiene un error en el código y tiene que localizarlo usando la herramienta de depuración vista en la sesión anterior. Para ello, ejecute el programa introduciendo el valor 40 y observe que no sale el 20 como divisor. Introduzca un punto de interrupción en el condicional que hay dentro del bucle y añada un watch para visualizar el valor de la variable divisor. Realice una ejecución paso a paso para localizar el error.

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

Sesión 5 (Tema II)

► Actividades a realizar en casa

Actividad: Lectura de programas resueltos.

Lea la solución de los siguientes ejercicios:

23 [Factorial y Potencia]

https://decsai.ugr.es/jccubero/FP/II_FactorialPotencia.cpp

Actividad: Resolución de problemas.

Se proporciona enlaces a videos que explican el esquema de la solución de algunos ejercicios. Consulte dichos videos **DESPUÉS** de haber intentado resolver el ejercicio por sus propios medios.

- **Obligatorios:**

Ejercicios sobre condicionales:

16 [Coronavirus]

https://decsai.ugr.es/jccubero/FP/II_Coronavirus.mp4

Ejercicios sobre bucles:

22 [Variación porcentual: lectura de varios valores]

https://decsai.ugr.es/jccubero/FP/II_VariacPorc.mp4

24 [Interés bancario (capital reinvertido)]

https://decsai.ugr.es/jccubero/FP/II_InterBancario.mp4

25 [Interés bancario (doblar)]

https://decsai.ugr.es/jccubero/FP/II_InterBancarioDoblar.mp4

27 [Número Narcisista]

https://decsai.ugr.es/jccubero/FP/II_Narcisista.mp4

28 [Mínimo de varios valores]

https://decsai.ugr.es/jccubero/FP/II_Min.mp4

- **Opcionales:**

26 [Tarifa aérea con filtro de entrada de datos]

- *Complementarios:*

Los siguientes ejercicios no tiene que entregarlos ni se defenderán. Si está empezando a programar le podrán ayudar a afianzar conceptos básicos.

20 [Intervalo: Lectura de valores]

29 [Velocidad imputada -lectura en bucle-]

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

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

Sesión 6 (Temas II y III)

Actividad:

Actividad: Lectura de programas resueltos.

Lea la solución de los siguientes ejercicios:

- Bucles simples:
30 [Aproximación de π por Gregory-Leibniz] - apartado a) -
https://decsai.ugr.es/jccubero/FP/II_AproximacionPiLeibniz.cpp
- Bucles anidados:
36 [Pirámide] y 37 [Cuadrado]
https://decsai.ugr.es/jccubero/FP/II_PiramideCuadrado.cpp

► Actividades a realizar en casa

Actividad: Resolución de problemas.

Se proporciona enlaces a videos que explican el esquema de la solución de algunos ejercicios. Consulte dichos videos **DESPUÉS** de haber intentado resolver el ejercicio por sus propios medios.

- **Obligatorios:**

- Ejercicios de la relación de Problemas II (bucles simples):

- 30 [Aproximación de π por Gregory-Leibniz] - apartados b) y c) -

- https://decsai.ugr.es/jccubero/FP/II_PiAproximacionLeibniz.mp4

- 31 [Aproximación de π por Wallis]

- https://decsai.ugr.es/jccubero/FP/II_PiAproximacionWallis.mp4

- 32 [Aproximación de π por Madhava sin usar pow]

- https://decsai.ugr.es/jccubero/FP/II_PiAproximacionMadhava.mp4

- Ejercicios de la relación de Problemas II (bucles anidados):

- 40 [Muy divisible]

- https://decsai.ugr.es/jccubero/FP/II_MuyDivisible.mp4

- 41 [Tarifa aérea: múltiples billetes]

- https://decsai.ugr.es/jccubero/FP/II_TarifaAereaMultiplesBilletes.mp4

- Ejercicios de la relación de Problemas III (Vectores):

- 1 [Divisores]

- https://decsai.ugr.es/jccubero/FP/III_Divisores.mp4

- **Opcionales:**

- 33 [Secuencia de temperaturas]

- https://decsai.ugr.es/jccubero/FP/II_SecuenciaTemperaturas.mp4

- **Complementarios:**

Los siguientes ejercicios no tiene que entregarlos ni se defenderán. Si está empezando a programar le podrán ayudar a afianzar conceptos básicos.

- 38 [Divisores anidado]

- Solución: https://decsai.ugr.es/jccubero/FP/II_DivisoresAnidado.cpp

39 [Interés bancario anidado]

Solución: https://decsai.ugr.es/jccubero/FP/II_InteresAnidado.cpp

► Actividades a realizar en las aulas de ordenadores

Redireccionando la entrada y salida de datos

Supongamos que tenemos un programa compilado `mi_programa.exe`. Podemos ejecutar el programa desde una consola del sistema y especificar que la entrada de datos será desde un fichero a través del símbolo de redireccionamiento `<` (no ha de confundirse con el token `<<` que aparecía en una instrucción `cout` de C++)

```
C:\Users\Carlos> mi_programa.exe < datos.txt
```

Cuando ejecutemos el programa, cada ejecución de `cin` leerá un dato desde el fichero de texto indicado, saltándose todos los espacios en blanco y tabuladores que hubiese previamente. Esta lectura es posible gracias al buffer de entrada de datos que vimos en la página 23. Este buffer se va llenando automáticamente con los datos del fichero.

Hay que destacar que este redireccionamiento de la entrada lo estamos haciendo en la llamada al ejecutable desde la consola del sistema operativo. También pueden leerse datos de un fichero desde dentro del propio código fuente del programa, pero esto se verá en el segundo cuatrimestre.

Para probarlo, descargue el fichero

https://decsai.ugr.es/jccubero/FP/II_cin.cpp

y guárdelo en una carpeta (por ejemplo `U:\FP\Redireccionamiento`) Compile el programa para obtener el ejecutable `II_cin.exe`. Descargue ahora el fichero

https://decsai.ugr.es/jccubero/FP/II_cin_datos_entrada.txt

y cópielo dentro de la misma carpeta en la que tiene el programa `II_cin.exe`.

Ahora, vamos a lanzar una ventana de consola del sistema operativo. Para ello, 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 cambiamos de directorio para situarnos en la carpeta en la que tenemos el ejecutable anterior. Por ejemplo:

```
C:\Documents and Settings\aulas> U:
```

```
U:> cd FP
```

```
U:\FP> cd Redireccionamiento
```

Introducimos la instrucción siguiente:

```
U:\FP\Redireccionamiento> II_cin.exe < II_cin_datos_entrada.txt
```

Cuando la lectura llegue al final del fichero, cualquier intento de otra lectura de datos producirá un *fallo*. En el segundo cuatrimestre se verá cómo controlar esta situación. Por ahora, en el primer cuatrimestre, cuando el número de datos a leer sea variable, simplificaremos la lectura de datos usando dos alternativas:

- O bien leemos al principio el número de datos n que se van a leer y luego se leen los n datos con un bucle.
- O bien leemos los datos hasta llegar a un *terminador*, que será un valor especial del mismo tipo de dato que estemos leyendo. Por ejemplo, leeremos caracteres hasta encontrar el carácter #, o leeremos enteros hasta llegar a -1. Este terminador será el último valor del fichero.

Al igual que hemos redirigido la entrada de datos, también se puede redirigir la salida de resultados para que ésta se haga sobre un fichero de texto. Basta usar el símbolo > seguido del nombre del fichero. En nuestro ejemplo anterior, sería:

```
II_cin.exe < II_cin_datos_entrada.txt > II_cin_datos_salida.txt
```

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

Sesión 7 (Temas II y III)

► **Actividades a realizar en casa**

Actividad: Lectura de programas resueltos.

Lea la solución de los siguientes ejercicios:

De la Relación de Problemas III:

2 [Palíndromo e Invierte]

https://decsai.ugr.es/jccubero/FP/III_PalindromoInvierte.cpp

7 [Comprobación Fecha]

https://decsai.ugr.es/jccubero/FP/III_Fecha.cpp

Actividad: Resolución de problemas.

Se proporciona enlaces a videos que explican el esquema de la solución de algunos ejercicios. Consulte dichos videos **DESPUÉS** de haber intentado resolver el ejercicio por sus propios medios.

- **Obligatorios:**

Ejercicios de la relación de Problemas III (Vectores):

4 [Sustituir carácter por vector (con vector auxiliar)]

https://decsai.ugr.es/jccubero/FP/III_SustituyeCaracterVector.mp4

5 [Sumatoria]

https://decsai.ugr.es/jccubero/FP/III_Sumatoria.mp4

6 [Frecuencias caracteres]

https://decsai.ugr.es/jccubero/FP/III_Frecuencias.mp4

13 [Contiene débil]

https://decsai.ugr.es/jccubero/FP/III_ContieneDebil.mp4

- **Opcionales:**

9 [Mayor desnivel]

https://decsai.ugr.es/jccubero/FP/III_MayorDesnivel.mp4

- **Complementarios:**

Los siguientes ejercicios no tiene que entregarlos ni se defenderán. Si está empezando a programar le podrán ayudar a afianzar conceptos básicos.

3 [Muy divisible con vectores]

8 [Cuenta Mayúsculas]

- **Complementarios avanzados:**

Los siguientes ejercicios no tiene que entregarlos ni se defenderán. Son ejercicios interesantes, típicos de un examen y que pueden resultarle útiles para preparar el examen de Fundamentos de Programación.

De la relación de Problemas II (bucles):

42 [Mayor nota media]

https://decsai.ugr.es/jccubero/FP/II_MayorNotaMedia.mp4

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

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

Sesión 8 (Tema III)

► Actividades a realizar en casa

Actividad: Lectura de programas resueltos.

Lea la solución de los siguientes ejercicios:

- Ejercicios sobre vectores:

10 [Moda]

https://decsai.ugr.es/jccubero/FP/III_Moda.cpp

16 [Elimina ocurrencias de una componente -versión ineficiente-]

https://decsai.ugr.es/jccubero/FP/III_EliminaOcurrenciasIneficiente.mp4

https://decsai.ugr.es/jccubero/FP/III_EliminaOcurrenciasIneficiente.cpp

- Sobre matrices (Tema III):

22 [Traspuesta]

https://decsai.ugr.es/jccubero/FP/III_Traspuesta.cpp

Actividad: Resolución de problemas.

Resuelva los siguientes ejercicios de la relación de problemas III:

- **Obligatorios:**

Ejercicios sobre vectores:

17 [Elimina ocurrencias de una componente -versión eficiente-]

https://decsai.ugr.es/jccubero/FP/III_EliminaOcurrenciasEficiente.mp4

Ejercicios sobre matrices:

23 [Producto de matrices]

https://decsai.ugr.es/jccubero/FP/III_ProductoMatrices.mp4

24 [Distancias entre ciudades]

https://decsai.ugr.es/jccubero/FP/III_Ciudades.mp4

25 [Distancias entre ciudades (mejor escala)]

https://decsai.ugr.es/jccubero/FP/III_Ciudades2.mp4

- **Opcionales:**

12 [Sistema de D'Hondt]

https://decsai.ugr.es/jccubero/FP/III_DHondt.mp4

- **Complementarios:**

Los siguientes ejercicios no tiene que entregarlos ni se defenderán. Si está empezando a programar le podrán ayudar a afianzar conceptos básicos.

- Ejercicios sobre vectores:

11 [Moda (versión 2)]

14 [Elimina un bloque (versión ineficiente)]

18 [Sustituir carácter por vector (versión ineficiente)]

https://decsai.ugr.es/jccubero/FP/III_SustituyeCaracterVectorIneficiente.cpp

- Ejercicios sobre matrices:

28 [Máximo de los mínimos]

- **Complementarios avanzados:**

Los siguientes ejercicios no tiene que entregarlos ni se defenderán. Son ejercicios interesantes, típicos de un examen y que pueden resultarle útiles para preparar el examen de Fundamentos de Programación.

15 [Elimina un bloque (versión eficiente)]

19 [Sustituir carácter por vector (versión eficiente)]

https://decsai.ugr.es/jccubero/FP/III_SustituyeCaracterVectorEficiente.mp4

Puede hacer los dos ejercicios siguientes cuando se haya visto en clase la ordenación de vectores:

20 [Top k (versión ineficiente)]

https://decsai.ugr.es/jccubero/FP/III_TopkIneficiente.mp4

21 [Top k (versión eficiente)]

https://decsai.ugr.es/jccubero/FP/III_Topk.mp4

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

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

Sesión 9 (Tema IV)

► **Actividades a realizar en casa**

Actividad: Lectura de programas resueltos.

Lea la solución de los siguientes ejercicios:

Sobre funciones (Tema IV):

2 [Potencia entera]

https://decsai.ugr.es/jccubero/FP/IV_FactorialPotencia.cpp

13 [Lee entero en rango]

https://decsai.ugr.es/jccubero/FP/IV_LeeIntRango.cpp

15 [Diseño de funciones sobre cadenas]

https://decsai.ugr.es/jccubero/FP/IV_DisenioFuncionesString.cpp

Actividad: Resolución de problemas.

Resuelva los siguientes ejercicios:

- **Obligatorios:**

Ejercicios sobre funciones:

1 [Errores en funciones]

10 [Máximo de 3 valores]

https://decsai.ugr.es/jccubero/FP/IV_Max.mp4

11 [Decimal redondeado]

https://decsai.ugr.es/jccubero/FP/IV_DecimalRedondeado.mp4

12 [Elasticidad precio-demanda]

https://decsai.ugr.es/jccubero/FP/IV_Elasticidad.mp4

14 [Lee entero mayor o igual que otro] (lea antes la solución del ejercicio 13 [Lee entero en rango] cuyo enlace se indica en la lectura de programas a leer para esta sesión)

https://decsai.ugr.es/jccubero/FP/IV_LeeIntMayor.mp4

16 [Uso de funciones sobre cadenas] (lea antes la solución del ejercicio 15 [Diseño de funciones sobre cadenas] cuyo enlace se indica en la lectura de programas a leer para esta sesión)

17 [DoubleToString]

https://decsai.ugr.es/jccubero/FP/IV_DoubleToString.mp4

- **Opcionales:**

Ejercicios sobre matrices:

26 [Asignación de pedidos a técnicos]

https://decsai.ugr.es/jccubero/FP/III_AsignacionPedidosTecnicos.mp4

- **Complementarios:**

Los siguientes ejercicios no tiene que entregarlos ni se defenderán. Si está empezando a programar le podrán ayudar a afianzar conceptos básicos.

Ejercicios sobre funciones:

3 [Valores de la Gaussiana]

4 [Mismo signo (multiplicando)]

Solución: https://decsai.ugr.es/jccubero/FP/IV_MismoSignoMultiplicando.cpp

5 [Interés bancario (capital reinvertido)]

Solución: https://decsai.ugr.es/jccubero/FP/IV_InteresReinvierte.cpp

6 [Interés bancario (doblar)]

- Complementarios avanzados:

Los siguientes ejercicios no tiene que entregarlos ni se defenderán. Son ejercicios interesantes, típicos de un examen y que pueden resultarle útiles para preparar el examen de Fundamentos de Programación.

Ejercicios sobre matrices:

27 [Matriz promedio]

29 [Máximo de los mínimos (eficiente)]

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

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

Sesión 10 (Tema IV)

► Actividades a realizar en casa

Actividad: Lectura de programas resueltos.

Lea la solución de los siguientes ejercicios:

7 [Grados a radianes]

https://decsai.ugr.es/jccubero/FP/IV_GradosToRadianes.cpp

Actividad: Resolución de problemas.

- *Obligatorios:*

Ejercicios sobre funciones void:

20 [Errores en funciones void]

21 [Mensaje inicial]

https://decsai.ugr.es/jccubero/FP/IV_Mensaje.mp4

Ejercicios sobre clases:

22 [Recta con datos miembro públicos]

23 [Dinero con datos miembro públicos]

https://decsai.ugr.es/jccubero/FP/IV_DineroPublic.mp4

- *Opcionales:*

18 [Tarifa aérea con funciones]

https://decsai.ugr.es/jccubero/FP/IV_TarifaAerea.mp4

- *Complementarios:*

Los siguientes ejercicios no tiene que entregarlos ni se defenderán. Si está empezando a programar le podrán ayudar a afianzar conceptos básicos.

8 [Diferencia entre instantes con funciones]

9 [Comparación entre instantes con funciones]

- Complementarios avanzados:

Los siguientes ejercicios no tiene que entregarlos ni se defenderán. Son ejercicios interesantes, típicos de un examen y que pueden resultarle útiles para preparar el examen de Fundamentos de Programación.

19 [Kaprekar]

Solución: https://decsai.ugr.es/jccubero/FP/IV_Kaprekar.cpp

► Actividades a realizar en las aulas de ordenadores

Depuración de funciones

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

Usaremos como ejemplo la función Combinatorio. En primer lugar, crearemos la carpeta IV_Combinatorio en la unidad local y copiaremos en ella el fichero fuente IV_Combinatorio.cpp disponible en:

https://decsai.ugr.es/jccubero/FP/IV_Combinatorio.cpp

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

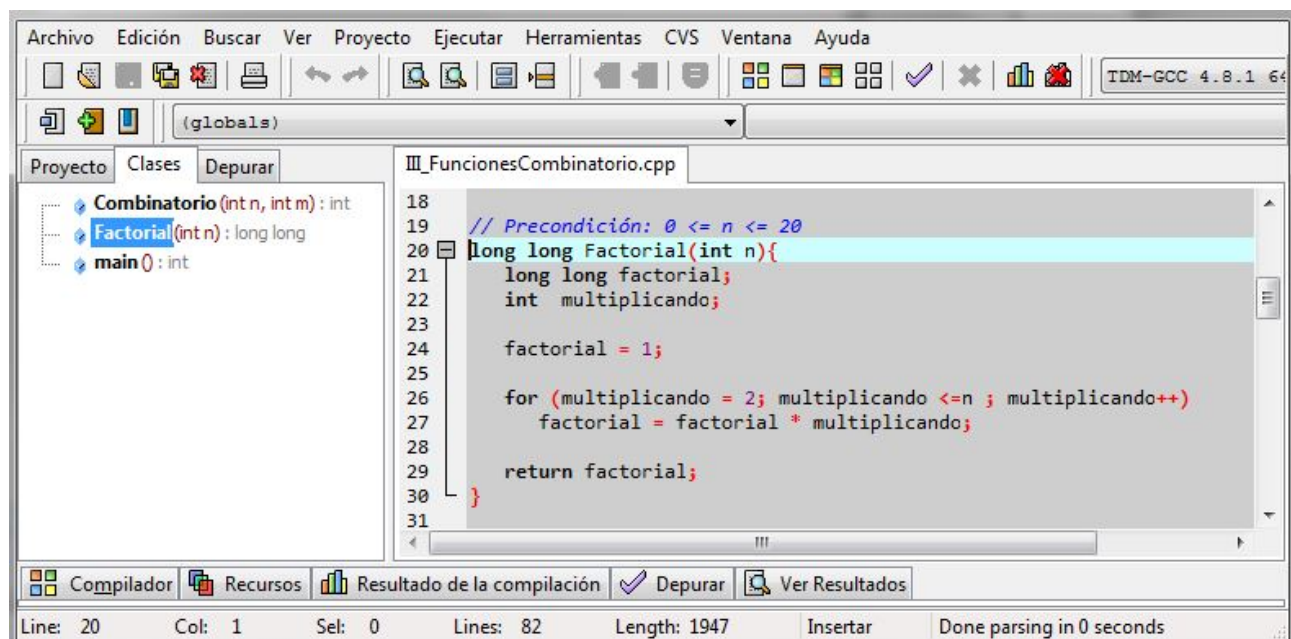


Figura 10: Explorador de clases

Para cada función muestra su *nombre*, los *parámetros formales* y su *tipo*, así como el *tipo de la función* (el tipo del valor devuelto). Se muestran todas las funciones en orden alfabético. Haciendo click sobre el nombre de cualquier función en el explorador, en el editor vemos

el código de la función seleccionada. Éste es una manera rápida de acceder al código de cualquier función en nuestros programas.

El proceso de depuración se inicia de la manera habitual:

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

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

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

y comenzamos la depuración.

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

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

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

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

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

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

En este punto,

- si se pulsa **Siguiente Paso** se completa la ejecución de esa línea y se cede el control a la línea siguiente. Observad que la variable local `combinatorio` contiene el valor ya calculado.
- si se pulsa **Avanzar Paso a paso** se entra a ejecutar la función `Factorial`, pasando el control a la primera instrucción de esa función.

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

Por último, suprima la línea en la que se hace el `return` en cualquiera de las funciones anteriores para que se familiarice con el mensaje de aviso que aparece al compilar el código. Observe que es un mensaje de aviso y no de error. El compilador realizará una acción con un comportamiento indeterminado cuando la función acabe sin ejecutar una sentencia `return`.

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

Sesión 11 (Tema IV)

► Actividades a realizar en casa

Actividad: Lectura de programas resueltos.

Lea la solución de los siguientes ejercicios:

24 [Recta con datos miembro privados]

https://decsai.ugr.es/jccubero/FP/IV_RectaPrivado.cpp

25 [Cronómetro]

https://decsai.ugr.es/jccubero/FP/IV_Cronometro.cpp

Actividad: Resolución de problemas.

Resuelva los siguientes ejercicios de la relación de problemas IV:

- **Obligatorios:**

26 [Dinero con datos miembro privados]

https://decsai.ugr.es/jccubero/FP/IV_Dinero.mp4

27 [Cuadrado]

https://decsai.ugr.es/jccubero/FP/IV_Cuadrado.mp4

28 [Cuadrado con constructor]

https://decsai.ugr.es/jccubero/FP/IV_CuadradoConstructor.mp4

29 [Recta con constructor]

30 [Generador aleatorio]

https://decsai.ugr.es/jccubero/FP/IV_GeneradorAleatorio.mp4

31 [Diseño de la interfaz de la clase SecuenciaCaracteres]

32 [Diseño de la interfaz de la clase Instante]

33 [Diseño de la interfaz de la clase FormateadorDoubles]

- **Opcionales:**

34 [Diseño de la interfaz de la clase SimuladorDeposito]

https://decsai.ugr.es/jccubero/FP/IV_SimuladorDeposito.mp4

35 [SimuladorDeposito (implementación)]

https://decsai.ugr.es/jccubero/FP/IV_SimuladorDeposito.mp4

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

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



Fundamentos de Programación.

Relaciones de Problemas.

© Copyright: Juan Carlos Cubero. Universidad de Granada.
Sugerencias: por favor, envíe un e-mail a
JC.Cubero@decsai.ugr.es

RELACIÓN DE PROBLEMAS DEL TEMA I. Introducción a C++

1. [Asignaciones secuenciales salario] Indique cuál sería el resultado de las siguientes operaciones:

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

salario_base = 1000;
incremento = 200;

salario_final = salario_base;
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. [Ley de Ohm] Cree un programa que pida un valor de intensidad y resistencia e imprima el voltaje correspondiente, según la *Ley de Ohm*:

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

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

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

$$\text{área círculo} = \pi r^2 \quad \text{longitud circunferencia} = 2\pi r$$

Se proponen las siguientes alternativas.

- a) Pedir al usuario que introduzca el valor de π
- b) Usar el literal 3.1416 donde sea necesario.

c) Usar un dato llamado π , asignarle el literal 3.1416 y usarlo donde sea necesario.

¿Cuál elegiría y por qué?

Finalidad: Uso de constantes. Dificultad Baja.

4. [Cálculo de π a partir del arco-seno] Si dividimos la longitud de cualquier circunferencia por el doble de su radio, se obtiene siempre el mismo número real, a saber 3.1415927... Este número es bien conocido y se designa por la letra griega π . Es irracional y por tanto tiene infinitas cifras decimales. Hay diversos métodos para obtener este valor, algunos mejores que otros. En posteriores ejercicios veremos algunos de ellos. En la página web https://en.wikipedia.org/wiki/Chronology_of_computation_of_pi se puede consultar los últimos logros en la obtención de nuevas cifras decimales de π .

Una forma de obtener π es a través de la función arco-seno, ya que $\frac{\pi}{6}$ es el ángulo en radianes de un seno de 0.5 (corresponde al seno de un ángulo de 30 grados):

$$\frac{\pi}{6} = \text{arco-seno}(0.5)$$

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

Recuerde que la función arco-seno está implementada en la función `asin` de la biblioteca `cmath`.

La salida debe ser 3.14159

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

5. [Variación porcentual] El *porcentaje de incremento* de una cantidad nos mide en qué proporción dicha cantidad ha experimentado un aumento. Por ejemplo, supongamos que el precio de un producto ha subido de 5 a 6 euros. En este caso, el porcentaje de incremento ha sido del 20 %. La fórmula matemática para obtenerlo es la siguiente:

$$100 \times \frac{\text{valor final} - \text{valor inicial}}{\text{valor inicial}}$$

En nuestro ejemplo, sería

$$100 \times \frac{6 - 5}{5} = 20$$

En el caso de que el valor final fuese menor que el valor inicial, se utiliza el término *porcentaje de decremento*. Se aplica la misma fórmula pero se cambia el signo para que el resultado sea positivo. Por ejemplo, si un producto baja de 6 a 5 euros, el porcentaje de decremento sería $\text{abs}(100(6 - 5)/6) = 16.6667\%$. Observe que las

cantidades involucradas son las mismas pero los porcentajes son distintos ya que influye quién es el valor inicial y final.

Para no hablar de aumentos o decrementos, en general, 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* es la función valor absoluto.

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.

Ejemplo de entrada: 5 6

— Salida correcta: *variación porcentual: 20%*

Ejemplo de entrada: 6 5

— Salida correcta: *variación porcentual: 16.6667%*

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

6. **[Subir sueldo]** 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 cómputo, multiplique por 1.02 el valor original. Para resolver este ejercicio tiene varias alternativas:
- a) Directamente hacer el cómputo `1.02 * salario_base` dentro de la sentencia `cout`
 - b) Introducir una variable `salario_final`, asignarle la expresión anterior y mostrar su contenido en la sentencia `cout`
 - c) Modificar la variable original `salario_base` con el resultado de incrementarla un 2%.

Indique qué alternativa elige y justifíquela.

Ejemplo de entrada: 30 — Salida correcta: 30.6

Ejemplo de entrada: 0 — Salida correcta: 0

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

7. **[Interés bancario]** Un banco presenta la siguiente oferta. Si se deposita una cantidad de euros dada por la variable `capital` durante un año a plazo fijo, se dará un interés dado por la variable `interes`. Realice un programa que lea una cantidad `capital` y un interés `interes` desde teclado. A continuación, el programa debe calcular en una

variable `total` el dinero que se tendrá al cabo de un año, aplicando la fórmula de abajo e imprimirá el resultado en pantalla.

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

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

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

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

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

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

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

Analice las ventajas o inconvenientes de hacerlo así.

Ejemplo de entrada: 300 5.4 — Salida correcta: 316.2

Ejemplo de entrada: 300 0 — Salida correcta: 300

Ejemplo de entrada: 0 5.4 — Salida correcta: 0

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

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

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

Debe calcular el cuadrado sin usar ninguna función de la biblioteca `cmath`. Para la raíz cuadrada, sí puede usar la función `sqrt`.

Ejemplo de entrada: 2.1 3.2 0.5 1.6 — Salida correcta: 2.26274

Ejemplo de entrada: 2.1 3.2 2.1 3.2 — Salida correcta: 0

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

9. [Conversión de grados a radianes] Queremos transformar una cantidad g dada en grados a radianes. Para ello, basta aplicar la siguiente fórmula:

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

Construya un programa que lea dos números reales que representarán dos grados. Debe calcular los radianes correspondientes e imprimir el resultado en pantalla.

Ejemplo de entrada: 20 90

— Salida correcta: 0.349066 1.5708

Ejemplo de entrada: 180 0

— Salida correcta: 3.14159 0

Para representar π utilice la fórmula indicada en el ejercicio 4 [Cálculo de π a partir del arco-seno] .

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

10. [Decimal redondeado] En `cmath` está definida la función `round` que permite redondear un real al entero más próximo. Por ejemplo:

`round(3.6)` devuelve 4

`round(3.5)` devuelve 4

`round(3.1)` devuelve 3

`round(3.49)` devuelve 3

Construya un programa para redondear un número real a cualquier cifra decimal. Por ejemplo:

El resultado de redondear 3.49 a la primera cifra decimal es 3.5

El resultado de redondear 3.49 a la segunda cifra decimal es 3.49

El resultado de redondear 3.496 a la segunda cifra decimal es 3.5

El programa principal leerá el número real y la posición de la cifra decimal e imprimirá el resultado. Puede usar la misma función `round` y la función `pow`, ambas incluidas en `cmath`. La función `pow` eleva un número a otro; por ejemplo, para elevar 3.5 a 7, basta usar la expresión `pow(3.5, 7)`

Ejemplo de entrada: 3.49 1 — Salida correcta: 3.5

Ejemplo de entrada: 3.49 2 — Salida correcta: 3.49

Ejemplo de entrada: 3.496 2 — Salida correcta: 3.5

Finalidad: Practicar la construcción de expresiones matemáticas. Dificultad Baja.

11. **[Tarifa aérea según km]** 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.

Utilice el tipo de dato `double` para todas las variables del programa. Este es un ejercicio docente; recuerde que, en un programa en producción, jamás usaremos tipos de datos con representación en coma flotante (como es el caso del tipo `double`) para representar una cantidad monetaria.

Ejemplo de entrada: 120 — Salida correcta: 162

Ejemplo de entrada: 200 — Salida correcta: 170

Ejemplo de entrada: 201 — Salida correcta: 170.1

Ejemplo de entrada: 452 — Salida correcta: 195.2

Finalidad: Trabajar con expresiones numéricas. Dificultad Baja.

12. **[Tarifa aérea con descuento]** Una compañía aérea quiere aplicar una política de descuentos al precio final de un billete. Por un lado, si el número de puntos del cliente es alto se le aplicará un descuento del 4%. Por otro lado, si es un trayecto largo, se le aplicará un descuento del 2%.

En este ejercicio sólo se le pide que construya un programa que lea el precio del billete y aplique e imprima el resultado de aplicar un descuento u otro. Los descuentos no se acumulan.

Utilice el tipo de dato `double` para todas las variables del programa.

Ejemplo de entrada: 300 — Salida correcta: 288 294

Ejemplo de entrada: 500 — Salida correcta: 480 490

Ejemplo de entrada: 0 — Salida correcta: 0 0

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

13. **[Segundos entre dos instantes]** Calcule el número de segundos que hay entre dos instantes del mismo día.

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

El programa leerá la hora, minuto y segundo del instante inicial y la hora, minuto y segundo del instante final (supondremos que los valores introducidos son correctos) y mostrará el número de segundos entre ambos instantes. Si el segundo instante es menor que el primero, se devolverá el número de segundos en negativo.

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

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

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

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

14. [Uso de constantes] Re-escriba las soluciones de los ejercicios 3 [Circunferencia] , 4 [Cálculo de π a partir del arco-seno] , 9 [Conversión de grados a radianes] , 11 [Tarifa aérea según km] y 12 [Tarifa aérea con descuento] usando constantes donde estime necesario.

Finalidad: Uso de constantes. Dificultad Baja.

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

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

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

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

Estaríamos tentados a escribir el siguiente código:

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

pero no funciona correctamente ¿Por qué?

Proponga una solución e impleméntela.

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

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

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

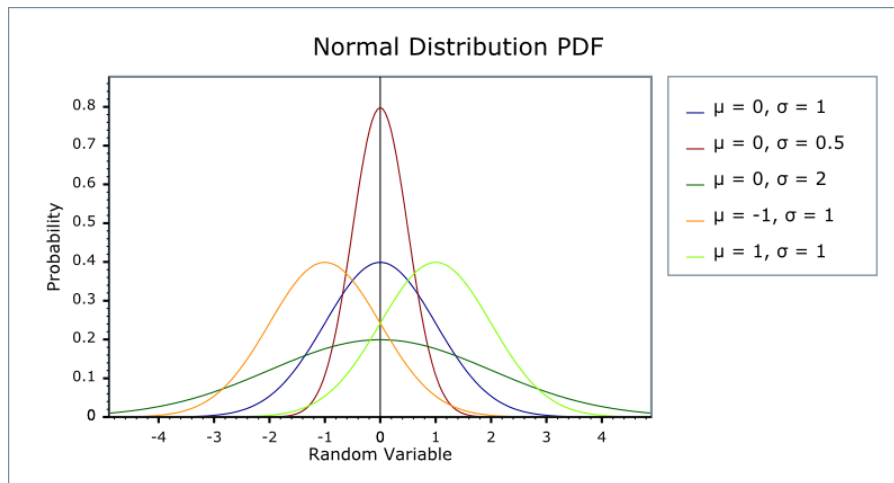
Finalidad: Plantear la solución de un ejercicio básico como es el de una conversión. Dificultad Baja.

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

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

$$\text{gaussiana}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\left(-0.5\left(\frac{x-\mu}{\sigma}\right)^2\right)}$$

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



Realice un programa que lea los coeficientes reales μ y σ de una función gaussiana. Dichos valores se conocen con el nombre de *esperanza* y *desviación* respectivamente. 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 `cmath`:

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

`pow(base, exponente)`

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

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

Compruebe que los resultados son correctos, usando la calculadora disponible en:

https://www.medcalc.org/manual/normal_distribution_functions.php

Rellene las cajas de texto que aparecen en PDFNormal en el orden siguiente: en primer lugar, el valor de x , a continuación la esperanza (μ) y por último la desviación típica (σ). Por ejemplo, para ver el valor de la función gaussiana con esperanza 12 y desviación 5 en el punto 2.5, debe poner en la calculadora online lo siguiente y a continuación hacer click en el signo =:

`PDFNormal(2.5, 12, 5) = 0.013123162955`

Tenga en cuenta que en el programa de C++ los datos se le piden en otro orden: primero la media y la desviación y luego el valor de la abscisa.

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

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

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

18. **[Intercambiar tres variables]** Se quiere generalizar el ejercicio 15 que intercambiaba el valor de dos variables al caso de tres variables. Construya un programa que declare las variables x , y , z , lea su valor desde teclado e intercambien entre sí sus valores de forma que el valor de x pasa a y , el de y pasa a z y el valor de z pasa a x (se pueden declarar variables auxiliares aunque se pide que se use el menor número posible).

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

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

19. **[Media y desviación]** Escriba un algoritmo para calcular la media aritmética muestral y la desviación estándar (o típica) muestral de las alturas de tres personas ($n=3$). Estos valores serán reales (de tipo `double`). La fórmula general para un valor arbitrario de n es:

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

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

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

Ejemplo de entrada: 160 170 180

— Salida correcta: Media: 170 Desviación: 8.16497

Ejemplo de entrada: 170 170 170

— Salida correcta: Media: 170 Desviación: 0

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

20. [Aproximación del valor de π] 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)

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

El resultado debe ser 3.16049, 3.125 y 3.14167

Finalidad: Mezclar tipos enteros y reales. Dificultad Baja.

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

Ejemplo de entrada: C — Salida correcta: 2

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

22. [Descuento tarifa aérea mezclando tipos] Recupere la solución del ejercicio 12 [Tarifa aérea con descuento]. En vez de fijar los dos descuentos del 4% y del 2% dentro del código, se quiere leer dichos valores con `cin`. Utilice el tipo de dato `int` para leer las dos variables correspondientes a los dos descuentos y el tipo `double` para el precio del billete.

Ejemplo de entrada: 300.4 4 2 — Salida correcta: 288.384 294.392

Ejemplo de entrada: 500 4 2 — Salida correcta: 480 490

Ejemplo de entrada: 0 4 2 — Salida correcta: 0 0

Finalidad: Mezclar tipos de datos numéricos en una expresión. Dificultad Baja.

23. **[Trunca decimales]** 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 truncamiento consiste en eliminar todos los decimales que hay a partir de n . El resultado debe guardarse en otra variable de tipo de dato `double`. Puede usar la función `trunc` de `cmath` (elimina la parte real de un número).

Ejemplo de entrada: 1.2349 2 — Salida correcta: 1.23
Ejemplo de entrada: 1.237 2 — Salida correcta: 1.23
Ejemplo de entrada: 1.237 1 — Salida correcta: 1.2
Ejemplo de entrada: 1.237 0 — Salida correcta: 1

Finalidad: Practicar el truncamiento de un real asignado a un entero. Dificultad Baja.

24. **[Lectura de un intervalo]** 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 como por ejemplo $(-\infty, \infty)$.

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

- Un carácter que represente el tipo de intervalo por la izquierda: el usuario deberá introducir el carácter ' $'$ ' si es abierto y ' $'$ ' si es cerrado.
- Un número real con la cota izquierda.
- El carácter ' $,$ ' como separador de las dos cotas.
- Un número real con la cota derecha.
- 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.

Ejemplo de entrada: (3.5 , 5.1] — Salida correcta: (3.5 , 5.1]

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

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

Ejemplo de entrada: D — Salida correcta: d

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

26. **[Pasar de carácter a entero]** Supongamos el siguiente código:

```
int entero;
char character;

cin >> character;
entero = character;
```

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

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

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

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

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

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

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

27. **[Expresiones lógicas]** En este ejercicio no puede usar estructuras condicionales (se ven en el Tema II). Como excepción, puede usar directamente literales, sin necesidad de definir constantes con los valores correspondientes.

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

Escriba una expresión lógica que sea verdadera si una variable de tipo entero llamada `edad` es mayor o igual que 18 o menor de 65 y falso en otro caso.

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

Escriba una expresión lógica que sea verdadera si un año es bisiesto y falso en otro caso. Los años bisiestos son aquellos que o bien son divisibles por 4 pero no por 100, o bien son divisibles por 400. Por ejemplo, son bisiestos: 1600, 1996, 2000, 2004. No son bisiestos: 1700, 1800, 1900, 1998, 2002.

Escriba una expresión lógica que sea verdadera si un dato de velocidad es mayor o igual que 100 kilómetros por hora y falso en otro caso.

Escriba una expresión lógica que sea verdadera si un carácter es una vocal (sólo se consideran vocales las minúsculas sin acentos) y falso en otro caso.

Escriba un programa que lea las variables `letra`, `edad`, `adivine`, `anio`, `velocidad` y `vocal`, calcule el valor de las expresiones lógicas almacenando el resultado en variables de tipo `bool` e imprima el resultado. Tenga en cuenta que cuando se imprime por pantalla (con `cout`) una variable lógica (o una expresión lógica en general) que es `true`, se imprime 1. Si es `false`, se imprime un 0. En el tema 2 veremos la razón.

Ejemplo de entrada: a 30 0 2017 120 p — Salida correcta: 1 1 0 0 1 0

Ejemplo de entrada: A 17 30 2000 90 e — Salida correcta: 0 0 1 1 0 1

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

28. [Elección tipo de dato] Indique qué tipo de dato usaría para representar:

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

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

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

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

```
a)    int chico, chico1, chico2;
      chico1 = 1234567;
      chico2 = 1234567;
      chico = chico1 * chico2;
```

- b)

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

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

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

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

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

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

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

Ejercicios complementarios

30. [Dos subidas de sueldo] Recupere la solución del ejercicio 6 [Subir sueldo] . Además de mostrar el salario con la subida del 2% se quiere mostrar el salario resultante de subirle otro 3% adicional. Esta segunda subida se realizará sobre el resultado de haber aplicado la primera subida. El programa debe mostrar los salarios resultantes (el resultante de la subida del 2% y el resultante de las dos subidas consecutivas del 2% y del 3%).

Ejemplo de entrada: 30 — Salida correcta: 30.6 31.518

Ejemplo de entrada: 0 — Salida correcta: 0 0

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

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

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

32. [Métricas atletismo] 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*).

Escriba dos programas para convertir entre estas dos medidas:

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

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

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

Importante: No repita cálculos ya realizados.

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

34. **[Reparto de ganancias mezclando tipos]** Realice el ejercicio del reparto de la ganancia de un producto, pero cambiando el tipo de dato de la ganancia total a `int` (el resto de variables siguen siendo `double`)

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

Dificultad Baja.

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

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

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

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

36. **[Pinta dígitos]** Escriba un programa que lea un valor entero e imprima en pantalla cada uno de sus dígitos separados por dos espacios en blanco. Supondremos que el usuario introduce siempre un entero de cinco dígitos, como por ejemplo 35197. En este caso, la salida sería:

3 5 1 9 7

Finalidad: Ejemplo de asignación acumulada.

Dificultad Media.

37. **[Población]** Los estudios poblacionales utilizan los conceptos de tasa de natalidad, mortalidad, etc. Al igual que un porcentaje representa una razón del total por cada cien (tanto por ciento), la tasa es una razón del total por cada mil (tanto por mil). Así pues una tasa de natalidad de 32, por ejemplo, significa que hay 32 nacimientos por cada 1000 habitantes.

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

Suponga que todos los datos son enteros.

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

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

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

38. **[Desplaza entero dentro de un intervalo]** Queremos construir una expresión numérica que desplace un entero un número 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 un 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`) le sumará el valor `desplazamiento` y lo convertirá en un entero dentro del intervalo `[minimo, maximo]` tal y como se ha descrito anteriormente.

Ejemplo de entrada: 65 90 3 70 — Salida correcta: 73

Ejemplo de entrada: 65 90 3 89 — Salida correcta: 66

Ejemplo de entrada: 65 90 3 90 — Salida correcta: 67

Finalidad: Trabajar con los operadores enteros. Dificultad Media.

39. **[Codificación de caracteres con algoritmo de rotación]** 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 las letras 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.

Para resolver este ejercicio sólo se pueden usar las herramientas de programación vistas en el primer tema. Se recomienda que revise la solución del ejercicio 38 **[Desplaza entero dentro de un intervalo]** de esta Relación de Problemas.

Ejemplo de entrada: 4 A — Salida correcta: E

Ejemplo de entrada: 4 Y — Salida correcta: C

Finalidad: Expresiones con caracteres y enteros. Dificultad Media.

40. **[Coordenadas geográficas]** En ejercicios posteriores, trabajaremos con un conjunto de coordenadas geográficas y necesitaremos realizar ciertas operaciones trigonométricas. En este ejercicio, vamos a implementar una parte de dichas operaciones.

Se pide construir un programa que lea cuatro variables reales grados_lat_1 , grados_lon_1 , grados_lat_2 , grados_lon_2 representando las coordenadas (en grados) de longitud y latitud de dos puntos en el plano y calcule el valor de la variable a aplicando la siguiente fórmula:

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

Los valores de lat_i y lon_i son los valores correspondientes a grados_lat_i grados_lon_i , pero expresados en radianes, por lo que tendrá que utilizar la conversión vista en el ejercicio 9 **[Conversión de grados a radianes]**

Tenga en cuenta que si escribe la expresión $1/2$ el resultado es cero ya que, al ser los operandos de tipo entero, el operador $/$ es la división entera.

Ejemplo de entrada: 37.18817 -3.60667 41.89193 12.51133

— Salida correcta: 0.0133395

Los datos anteriores corresponden a la latitud y longitud de Granada y Roma respectivamente.

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

41. **[Coordenadas geográficas (distancia)]** Recupere la solución del ejercicio 40 **[Coordenadas geográficas]** de la Relación de Problemas I.

Se quiere calcular la distancia entre dos puntos de la Tierra. Cada punto vendrá determinado por sus coordenadas geográficas, dadas por tres datos reales: su longitud, latitud y altura. Lo vamos a hacer de dos formas:

- a) Sin tener en cuenta la altura. La distancia *sobre plano* entre dos puntos se calcula como la longitud del segmento d_p de la figura y se puede calcular con la llamada *fórmula del Haversine*:

I) Se calcula $\Delta_{\text{lon}} = \text{lon}_2 - \text{lon}_1$ y $\Delta_{\text{lat}} = \text{lat}_2 - \text{lat}_1$

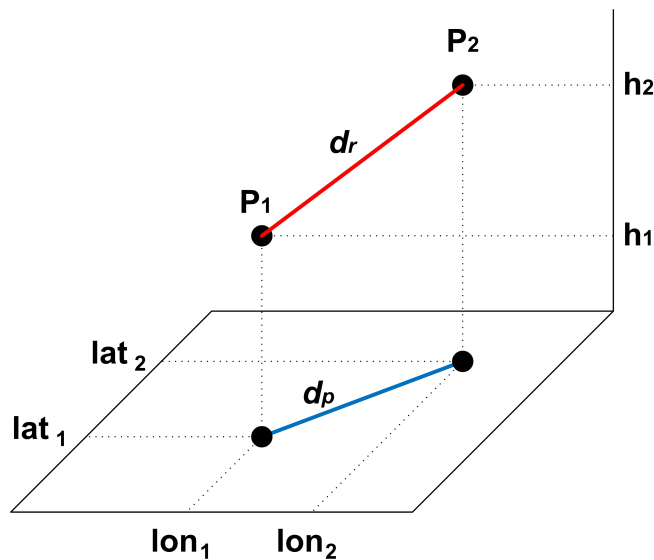
II) Se calcula $a = \sin^2 \left(\frac{1}{2} \Delta_{\text{lat}} \right) + \cos(\text{lat}_1) \cos(\text{lat}_2) \sin^2 \left(\frac{1}{2} \Delta_{\text{lon}} \right)$

III) Se calcula $c = 2 \arcsin(\min(1.0, \sqrt{a}))$ donde $\min(1, \sqrt{a})$ representa el mínimo entre 1 y \sqrt{a} .

- iv) Finalmente, la distancia será $d_p = R c$, donde $R = 6372797.560856$ metros es la longitud media del radio terrestre.

Tenga en cuenta lo siguiente:

- i) Los datos de latitud y longitud en las fórmulas, vienen expresados en radianes.
 - ii) La función arco-seno (`arcsin`) viene ya implementada en la biblioteca `cmath` con el nombre `asin`
 - iii) `mín` representa el mínimo de dos valores. Para implementarlo utilice la función `min` que viene definida en la biblioteca `algorithm`.
- b) Teniendo en cuenta la altura se obtiene la distancia *real* entre los dos puntos. Dicha distancia es la longitud del segmento d_r en la figura. Para calcular dicho valor, basta observar que se puede formar un triángulo rectángulo a partir de los segmentos d_p , d_r y la diferencia de las alturas (representadas por h_1 y h_2 en el gráfico) y aplicar el teorema de Pitágoras. La longitud del segmento d_p es la distancia euclídea entre los extremos del segmento.



Construya un programa que lea las coordenadas del primer punto (latitud, longitud y altura), las del segundo e imprima en pantalla la distancia sobre plano y la distancia real. Supondremos que las coordenadas de los puntos se introducen en grados, por lo que habrá que pasarlas a radianes para poder aplicar las fórmulas anteriores, tal y como hicimos en el ejercicio 9 [Conversión de grados a radianes] de esta Relación de Problemas.

Ejemplo de entrada: 37.18817 -3.60667 738 41.89193 12.51133 52
— Salida correcta: 1475367.200551 1475367.36003946

Los datos anteriores corresponden a la latitud, longitud y altura de Granada y Roma respectivamente.

Para poder ver los reales con varios dígitos de precisión, incluya la sentencia `cout.precision(15);` al inicio del programa, por ejemplo, después de declarar las variables.

Finalidad: Trabajar con expresiones numéricas. Dificultad Baja.

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

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

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

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

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

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

Dificultad Baja.

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

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

Buscad en Internet el por qué de dicha fórmula.

Dificultad Baja.

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

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

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

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

Algunas funciones de `cmath`

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

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

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

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

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

Dificultad Baja.

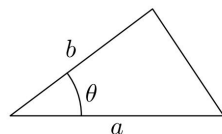
47. Dos locomotoras parten de puntos distintos avanzando en dirección contraria sobre la misma vía. Se pide redactar un programa para conocer las distancias que habrán recorrido ambas locomotoras antes de que choquen teniendo en cuenta que la primera locomotora viaja a una velocidad constante V_1 , que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

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

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

Dificultad Baja.

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



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

Dificultad Baja.

49. Los compiladores utilizan siempre el mismo número de bits para representar un tipo de dato entero (este número puede variar de un compilador a otro). Por ejemplo, 32

bits para un `int`. Pero, realmente, no se necesitan 32 bits para representar el 6, por ejemplo, ya que bastarían 3 bits:

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

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

Dificultad Media.

RELACIÓN DE PROBLEMAS DEL TEMA II. Estructuras de Control

Ejercicios sobre condicionales

1. **[Mismo signo -multiplicando-]** Construya un programa que lea dos datos a y b de tipo `int` y nos diga si tienen el mismo signo. Se considera que el cero no tiene signo por lo que cualquier número (incluido el cero) tiene un signo distinto del cero.

Para ello, basta comprobar si la multiplicación de a y b es positiva o no.

Ejemplo de entrada: 0 0 — Salida correcta: No tienen el mismo signo

Ejemplo de entrada: 0 3 — Salida correcta: No tienen el mismo signo

Ejemplo de entrada: 3 0 — Salida correcta: No tienen el mismo signo

Ejemplo de entrada: 3 4 — Salida correcta: Tienen el mismo signo

Ejemplo de entrada: -3 -4 — Salida correcta: Tienen el mismo signo

Ejemplo de entrada: 3 -4 — Salida correcta: No tienen el mismo signo

Ejemplo de entrada: -3 4 — Salida correcta: No tienen el mismo signo

Finalidad: Trabajar con estructuras condicionales dobles. Dificultad Baja.

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

Ejemplo de entrada: 3 4 — Salida correcta: Ninguno divide al otro

Ejemplo de entrada: 2 4 — Salida correcta: Uno de ellos divide al otro

Ejemplo de entrada: 4 2 — Salida correcta: Uno de ellos divide al otro

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

3. **[Año bisiesto]** Recuerde la definición de año bisiesto vista en el ejercicio 27 **[Expresiones lógicas]** de la Relación de Problemas I. Construya un programa para que lea un año y nos diga si es o no bisiesto con un mensaje en pantalla.

Ejemplo de entrada: 2017 — Salida correcta: No es bisiesto

Ejemplo de entrada: 2000 — Salida correcta: Es bisiesto

Finalidad: Plantear una estructura condicional simple con salida de resultados. Dificultad Baja.

4. [Tarifa aérea] Retome la solución del ejercicio 11 [Tarifa aérea según km] de la Relación de Problemas I disponible en:

https://decsai.ugr.es/jccubero/FP/I_TarifaAerea.cpp

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 301).

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

Ejemplo de entrada: 120 — Salida correcta: 150

Ejemplo de entrada: 300 — Salida correcta: 150

Ejemplo de entrada: 301 — Salida correcta: 150.1

Ejemplo de entrada: 452 — Salida correcta: 165.2

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

5. [Comparación de dos instantes (restando segundos)] Construya un programa que lea dos instantes de tiempo (según se define en el ejercicio 13 [Segundos entre dos instantes] de la Relación de Problemas I) y nos diga si el primero es anterior al segundo. Para ello, calcule los segundos que hay de diferencia entre ambos instantes, tal y como se hizo en el ejercicio 13 [Segundos entre dos instantes] de la Relación de Problemas I.

Ejemplo de entrada: 9 12 9 10 34 55

— Salida correcta: El primero es anterior

Ejemplo de entrada: 10 34 55 9 12 9

— Salida correcta: El primero no es anterior

Ejemplo de entrada: 10 34 55 10 34 55

— Salida correcta: El primero no es anterior

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

6. [Operadores lógicos] En este ejercicio no hace falta construir ningún programa. Debe crear un fichero de texto (con extensión cpp o txt) explicando qué problemas observa en los siguientes condicionales:

```
a)    char tipo_radar;
      cin >> tipo_radar;

      if (tipo_radar == 'F' && tipo_radar == 'f')
          .....
```

```
b)    double velocidad;
      cin >> velocidad;

      if (velocidad > 100 && velocidad < 70)
          cout << "\nVelocidad fuera del rango";

c)    double velocidad;
      cin >> velocidad;

      if (velocidad > 100 || velocidad > 110)
          cout << "Velocidad excesiva";
```

Finalidad: Entender cómo funcionan los operadores lógicos. Dificultad Baja.

7. [Tarifa aérea con descuentos] Recupere la solución del ejercicio 4 [Tarifa aérea] de esta Relación de Problemas. Una vez se ha obtenido el precio del billete (con el incremento de 10 céntimos por km recorrido, en su caso), se quieren aplicar varios descuentos **acumulativos**.

Un primer descuento vendrá determinado por el número de kilómetros del trayecto. El segundo descuento dependerá del número de puntos de la tarjeta de fidelización del cliente que ha comprado el billete. En concreto:

- a) 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 %.
- b) Si el trayecto es mayor de 700 km, se aplica un descuento del 2 %

Los dos descuentos anteriores (por longitud del trayecto y el correspondiente al número de puntos), son independientes y acumulables. En cualquier caso, ambos se aplican sobre el precio del billete (una vez le ha aumentado el recargo por el número de kilómetros), es decir, un descuento no se aplica sobre el resultado de haber aplicado previamente el otro descuento. Por lo tanto, un cliente podría beneficiarse de un descuento del 2%, del 3%, del 4%, del 5%, del 6%, o ninguno.

Construya un programa que lea el número de kilómetros del trayecto y el número de puntos del cliente e imprima en pantalla el precio final del billete con los descuentos aplicados, en su caso.

Ejemplo de entrada: 200 90 — Salida correcta: 150
Ejemplo de entrada: 200 120 — Salida correcta: 145.5
Ejemplo de entrada: 200 250 — Salida correcta: 144
Ejemplo de entrada: 650 90 — Salida correcta: 185
Ejemplo de entrada: 650 120 — Salida correcta: 179.45
Ejemplo de entrada: 800 90 — Salida correcta: 196
Ejemplo de entrada: 800 120 — Salida correcta: 190
Ejemplo de entrada: 800 250 — Salida correcta: 188

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

8. **[Tabulación]** . Indique si los siguientes trozos de código están correctamente tabulados según las normas indicadas en clase. En caso de que no lo estén, re-escribalos:

```
a) if (edad > 18)
    es_mayor_edad = true;
    else es_mayor_edad = false;

b) if (edad > 18)
    if (ingresos <= MAX_INGRESOS)
        suscept_ayuda = true;
    else suscept_ayuda = false;
    else if (ingresos > MIN_INGRESOS)
        suscept_ayuda = true;
    else
        suscept_ayuda = false;
```

Finalidad: Hacer énfasis en la importancia de la correcta tabulación del código. Dificultad Baja.

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

Ejemplo de entrada: D — Salida correcta: d

Ejemplo de entrada: d — Salida correcta: d

Ejemplo de entrada: ! — Salida correcta: !

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

10. **[Pasar de mayúscula a minúscula y viceversa]** Queremos modificar el ejercicio 9 para leer un carácter `letra_original` desde teclado y hacer lo siguiente:
- Si es una letra mayúscula, almacenaremos en la variable `letra_convertida` la correspondiente letra minúscula.
 - Si es una letra minúscula, almacenaremos en la variable `letra_convertida` la correspondiente letra mayúscula.
 - Si es un carácter no alfabético, almacenaremos el mismo carácter en la variable `letra_convertida`

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

Ejemplo de entrada: D — Salida correcta: d

Ejemplo de entrada: d — Salida correcta: D

Ejemplo de entrada: ! — Salida correcta: !

Finalidad: Plantear una estructura condicional anidada. Dificultad Baja.

11. **[Mayoría absoluta]** Se quiere construir un programa que calcule las posibles coaliciones de dos partidos que se pueden realizar para alcanzar la mayoría absoluta después de una votación en la que únicamente participan 3 partidos. Cuando se vea el tema de vectores se ampliará este ejercicio para tratar con un número cualquiera de partidos.

El programa leerá el número de votos presenciales que ha obtenido el primer partido (llamémosle A) y a continuación leerá el número de votos por correo obtenidos por el mismo partido. A continuación se leerán los mismos datos de otros dos partidos más (llamémosles B y C). El programa debe mostrar si hay algún partido que ha alcanzado la mayoría absoluta, es decir, que el total de votos obtenidos (los presenciales más los obtenidos por correo) sea mayor estricto que la mitad de la suma total de votos de todos los partidos. En el caso de que ningún partido haya alcanzado la mayoría absoluta, el programa debe mostrar las coaliciones de dos partidos que lleven a una mayoría absoluta. Debe tener en cuenta que puede haber más de una posible coalición que obtenga dicha mayoría absoluta.

Si se desea, puede usar el esqueleto del programa disponible en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/II_MayoriaAbsolutaEsqueleto.cpp

Por simplificar, se utilizarán enteros pequeños en los casos de prueba:

Ejemplo de entrada: 2 4 3 3 12 13

— Salida correcta: Mayoría absoluta: C

Ejemplo de entrada: 2 4 3 3 5 1

— Salida correcta: Mayoría absoluta: AB AC BC

Ejemplo de entrada: 0 0 0 0 0 0

— Salida correcta: Mayoría absoluta: Error en los datos

Ejemplo de entrada: 1 0 2 2 2 1

— Salida correcta: Mayoría absoluta: AB BC

Finalidad: Analizar si se han de usar estructuras condicionales anidadas o consecutivas. Dificultad Baja.

12. **[Velocidad imputada]** La velocidad de los vehículos en las vías de circulación es captada por unos aparatos electrónicos conocidos como radares o *cinemómetros*. Éstos pueden ser fijos o móviles y presentan un margen de error que hay que aplicar a la

velocidad captada por el radar. Dichos márgenes 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. La aplicación se realiza en el sentido más ventajoso para el contribuyente, es decir, disminuyendo la velocidad captada. 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 $104 - 5\% \text{ de } 104 = 104 - 5.2 = 98.8$ km/h en el caso de un radar fijo y de $104 - 7\% \text{ de } 104 = 104 - 7.28 = 96.72$ 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.

Ejemplo de entrada: F 95 — Salida correcta: 90

Ejemplo de entrada: M 95 — Salida correcta: 88

Ejemplo de entrada: F 104 — Salida correcta: 98.8

Ejemplo de entrada: M 104 — Salida correcta: 96.72

Nota:

En el ejercicio 13 [Multa autovía] de esta Relación de Problemas, se usa la velocidad del vehículo para establecer la sanción correspondiente en una autovía (dicho ejercicio no tiene que resolverlo).

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

13. [Multa autovía] 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. 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)

121	->	0,	100
151	->	2,	300
171	->	4,	400
181	->	6,	500
191	->	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.

14. [Comparación de dos instantes] Realice lo mismo que se le pide en el ejercicio 5 [Comparación de dos instantes (restando segundos)] pero resolviéndolo de otra forma. En vez de calcular los segundos que hay de diferencia entre ambos instantes, tiene que usar los valores de la hora, minuto y segundo del instante inicial y los tiene que comparar (usando expresiones lógicas) con los del instante final.

Ejemplo de entrada: 9 12 9 10 34 55
— Salida correcta: El primero es anterior
Ejemplo de entrada: 10 34 55 9 12 9
— Salida correcta: El primero no es anterior
Ejemplo de entrada: 10 34 55 10 34 55
— Salida correcta: El primero no es anterior

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

15. [Intervalo] Recupere la solución del ejercicio 24 [Lectura de un intervalo] de la Relación de Problemas I. Puede usar la solución disponible en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/I_Intervalo.cpp

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.

Ejemplo de entrada: (3.5 , 5.1] 4.8
— Salida correcta: El valor 4.8 está dentro del intervalo (3.5, 5.1]
Ejemplo de entrada: (3.5 , 5.1] 8.3
— Salida correcta: El valor 8.3 no está dentro del intervalo (3.5, 5.1]
Ejemplo de entrada: (3.5 , 5.1] 3.5
— Salida correcta: El valor 3.5 no está dentro del intervalo (3.5, 5.1]
Ejemplo de entrada: (3.5 , 5.1] 5.1
— Salida correcta: El valor 5.1 está dentro del intervalo (3.5, 5.1]

Finalidad: Plantear una condición compleja. Dificultad Media.

16. [Coronavirus] Por todos es conocido el gran daño humano y económico que ha producido el coronavirus SARS-CoV-2 al provocar la enfermedad denominada COVID-19. En Argentina, la web gubernamental <https://coronavirus.argentina.gob.ar/> diseñó un test aproximado para su identificación. El código fuente estaba escrito en JavaScript y parte de él se muestra en la imagen del siguiente enlace (como puede apreciar, Java es un lenguaje con una sintaxis similar a C++)

https://decsai.ugr.es/jccubero/FP/II_coronavirus.png

La lógica implementada en este código es correcta y por tanto identifica adecuadamente la enfermedad del COVID-19. Sin embargo, este código tiene varios problemas (a los pocos días de publicarlo arreglaron parte de ellos) en el sentido de que incumple algunas normas que un buen programador ha de seguir. Uno de los problemas más sencillos de detectar es la aparición duplicada de las comprobaciones sobre `respiratoryDisease`. En cualquier caso, para simplificar el problema nos vamos a fijar únicamente en cuatro síntomas, a saber, `bodyTemperature`, `difficultyBreathing`, `diabetes` y `cancer`, de forma que el código anterior se simplificaría en el siguiente:

```
if((bodyTemperature >= 38 && difficultyBreathing) ||
    (bodyTemperature >= 38 && difficultyBreathing && diabetes) ||
    (bodyTemperature >= 38 && difficultyBreathing && cancer) ||
    (bodyTemperature >= 38 && diabetes) ||
    (bodyTemperature >= 38 && cancer))
    cout << "Consulte autoridades locales";
else
    if (bodyTemperature >= 38)
        cout << "Cuarentena en su casa";
    else
        if (bodyTemperature < 38)
            cout << "Test negativo";
        else
            cout << "Test negativo";
```

Identifique los problemas que pueda haber y proponga una solución modificando el código del programa que puede encontrar en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/II_CoronavirusEsbozo.cpp

Ejemplo de entrada: 37 S S S

— Salida correcta: Test negativo

Ejemplo de entrada: 39 S N N

— Salida correcta: Consulte autoridades locales

Ejemplo de entrada: 39 N N N

— Salida correcta: Cuarentena en su casa

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

Finalidad: Separación de E/S y C. Dificultad Baja.

17. [Mismo signo con condicionales] Recupere la solución del ejercicio 1 [Mismo signo -multiplicando-] de esta Relación de Problemas. Resuelva el mismo problema de ver si dos enteros tienen el mismo signo sin utilizar la operación de multiplicación. Para ello, se pide que implemente las siguientes posibilidades:

a) Enumerando todas las posibles situaciones, es decir, los dos positivos, los dos negativos, el primero positivo y el segundo negativo o el primero negativo y el segundo positivo.

Si resolvemos de esta forma el problema, el programa será muy propenso a errores. ¿Por qué?

b) Utilice una estructura condicional anidada para no tener que enumerar todos los casos posibles.

c) Construya una única expresión lógica y asígnele su valor a una variable lógica.

Ejemplo de entrada: 0 0 — Salida correcta: No tienen el mismo signo

Ejemplo de entrada: 0 3 — Salida correcta: No tienen el mismo signo

Ejemplo de entrada: 3 0 — Salida correcta: No tienen el mismo signo

Ejemplo de entrada: 3 4 — Salida correcta: Tienen el mismo signo

Ejemplo de entrada: -3 -4 — Salida correcta: Tienen el mismo signo

Ejemplo de entrada: 3 -4 — Salida correcta: No tienen el mismo signo

Ejemplo de entrada: -3 4 — Salida correcta: No tienen el mismo signo

Finalidad: Trabajar con estructuras condicionales anidadas y expresiones lógicas compuestas. Dificultad Media.

18. [Mayúscula a minúscula y viceversa usando un enumerado] Modifique la solución al ejercicio 10 [Pasar de mayúscula a minúscula y viceversa] cuyo código puede descargar del siguiente enlace:

https://decsai.ugr.es/jccubero/FP/II_Mayuscula.cpp

para que, dependiendo de cómo era la letra introducida, imprima en pantalla alguno de los siguientes mensajes:

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

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

¿Cuál sería el inconveniente de usar dos variables de tipo bool?

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

Ejercicios sobre bucles

19. **[Divisores de un entero]** Realice un programa que lea desde teclado un entero e imprima en pantalla todos sus divisores positivos propios. Recuerde que los divisores propios de un entero n son los divisores de n distintos de 1 y el mismo n . Para obtener los divisores, basta recorrer todos los enteros menores que el valor introducido y comprobar si lo dividen. A continuación, mejorar el ejercicio para que la lectura del entero se realice usando un filtro con un bucle post test (`do while`), obligando a que sea positivo.

En este ejercicio, como en muchos otros de esta Relación de Problemas, tendrá que mezclar entradas y salidas de datos con los cálculos. En el tema III, cuando se vean los vectores, podrá separar ambas tareas.

Ejemplo de entrada: 16 — Salida correcta: 2, 4, 8

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

20. **[Intervalo: Lectura de valores]** Recupere la solución del ejercicio 15 **[Intervalo]** de esta 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.

Ejemplo de entrada: (3.5 , 5.1] 2 4.8 8.3
— Salida correcta:

El valor 4.8 está dentro del intervalo (3.5, 5.1]

El valor 8.3 no está dentro del intervalo (3.5, 5.1]

Finalidad: Plantear una estructura repetitiva básica con un condicional dentro. Dificultad Baja.

21. **[Valores de la Gaussiana]** Recupere la solución del ejercicio 17 **[Gaussiana]** de la relación de problemas I. Puede usar la solución propuesta en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/I_Gaussiana.cpp

Se pide construir un programa para imprimir el resultado de aplicar dicha función a varios valores de abscisas.

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

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

es decir, mínimo, mínimo + incremento, mínimo + 2*incremento, ..., hasta llegar, como mucho, a máximo. El incremento ha de leerse con un filtro de entrada para obligar a que sea estrictamente positivo.

```
12    <- Esperanza  (sin restricción)
5     <- Desviación (>= 0)
2     <- Mínimo     (sin restricción)
3     <- Máximo     (>= Mínimo)
0.5   <- Incremento (> 0)
```

Ejemplo de entrada: 12 5 2 3 0.5

— Salida correcta: 0.0107982 0.0131232 0.01579

Ejemplo de entrada: 12 -5 5 2 3 0.5

— Salida correcta: 0.0107982 0.0131232 0.01579

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

— Salida correcta: 0.0107982 0.0131232 0.01579

Finalidad: Ejemplo básico de bucle. Dificultad Baja.

22. **[Variación porcentual: lectura de varios valores]** Recupere la solución del ejercicio 5 **[Variación porcentual]** de la Relación de Problemas I. Modifíquelo para realizar una lectura de múltiples pares de valores. La entrada de datos se interrumpirá cuando se introduzca cualquier valor negativo. Para simplificar, supondremos que si el primer valor introducido es positivo, el usuario también introducirá un positivo como segundo valor.

Por cada par de valores, el programa mostrará la correspondiente variación porcentual. En este ejercicio puede mezclar entradas de datos con salidas y cálculos, dentro del mismo bucle.

Ejemplo de entrada: 5 6 6 5 -1

— Salida correcta: Variaciones porcentuales: 20% 16.6667%

Finalidad: Controlar la lectura de pares de valores con terminador. Dificultad Baja.

23. **[Factorial y Potencia]** Calcule mediante un programa en C++ la función potencia x^n , y la función factorial $n!$ con n un valor entero y x un valor real. No pueden usarse las funciones definidas en `cmath`, por lo que tendrá que implementar los cálculos con los bucles necesarios.

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

$$0! = 1$$

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

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

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

24. [Interés bancario (capital reinvertido)] Recupere la solución del ejercicio 7 [Interés bancario] de la Relación de Problemas I. Puede usar el código disponible en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/I_InterBancario.cpp

Supongamos ahora que se quiere reinvertir todo el dinero obtenido (el original C más los intereses producidos) en otro plazo fijo a un año y así, sucesivamente. Construya un programa para que lea el capital, el interés I y un número de años N , y calcule e imprima todo el dinero obtenido durante cada uno de los N años, suponiendo que todo lo ganado (incluido el capital original C) se reinvierte a plazo fijo durante el siguiente año.

Para resolver este ejercicio debe plantear una estructura repetitiva. Si lo desea, puede comprobar que el resultado es correcto, comparándolo con el de la fórmula del interés compuesto que obtiene directamente el capital final acumulado:

$$C * (1 + I/100)^N$$

Ejemplo de entrada: 300 5.4 3

— Salida correcta:

```
Capital obtenido transcurrido el año número 0 = 316.2
Capital obtenido transcurrido el año número 1 = 333.275
Capital obtenido transcurrido el año número 2 = 351.272
```

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

25. [Interés bancario (doblar)] Recupere la solución del ejercicio 7 [Interés bancario] de la Relación de Problemas I. Construya un programa para calcular cuántos años han de pasar hasta llegar a doblar, como mínimo, el capital inicial. Los datos que han de leerse desde teclado son el capital inicial y el interés anual.

Ejemplo de entrada: 300 5.4

— Salida correcta:

```
Para doblar la cantidad inicial han de pasar 14 años
Al finalizar, se obtendrá un total de 626.455 euros
```

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

26. [Tarifa aérea con filtro de entrada de datos] Recupere la solución del ejercicio 7 [Tarifa aérea con descuentos] de esta Relación de Problemas. Puede encontrar una solución en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/II_TarifaAereaDescuentos.cpp

Modifíquelo para obligar al usuario a introducir valores correctos. Por lo tanto, debe usar un filtro de entrada de datos para que el número de kilómetros sea mayor o igual que cero y el número de puntos sea un valor entre 0 y 400 (máximo número posible de puntos).

En el siguiente ejemplo, observe que los filtros de entrada de datos *rechazan* los datos -2 -5 (el número de kms debe ser positivo) y 1300 -450 (el número de puntos ha de estar entre 0 y 400).

Ejemplo de entrada: -2 -5 200 1300 -450 250 — Salida correcta: 144

Finalidad: Uso de bucles en filtros de entrada de datos con condiciones compuestas. Dificultad Baja.

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

Ejemplo de entrada: 153 — Salida correcta: El número es narcisista

Ejemplo de entrada: 152 — Salida correcta: El número no es narcisista

Finalidad: Ejercitar los bucles. Dificultad Media.

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

Ejemplo de entrada: 0

— Salida correcta: Introducidos: 0. Mínimo: 0

Ejemplo de entrada: 1 3 -1 2 0

— Salida correcta: Introducidos: 4. Mínimo: -1

Ejemplo de entrada: 1 3 1 2 0

— Salida correcta: Introducidos: 4. Mínimo: 1

Una vez hecho el programa, indique qué cambios debería realizar si los valores a leer son enteros negativos y el final de la entrada de datos lo marca la introducción de cualquier valor positivo.

Ejemplo de entrada: -1 -3 -2 6

— Salida correcta: Introducidos: 3. Mínimo: -3

RELACIÓN DE PROBLEMAS DEL TEMA II. Estructuras de Control

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

29. [Velocidad imputada -lectura en bucle-] Recupere la solución del ejercicio 12 [Velocidad imputada] de esta Relación de Problemas. Modifique el programa principal 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 los datos leídos son correctos (siempre serán 'F' o 'M'). El programa calculará las velocidades imputadas según se indicó en el ejercicio 12 [Velocidad imputada] e imprimirá en pantalla la matrícula de aquel vehículo que tenga máxima velocidad imputada (también imprimirá dicha velocidad). La lectura de los datos terminará cuando la matrícula del vehículo sea la cadena "#".
- Ejemplo de entrada: 4312BHG F 95 8342DFW F 104 4598IJG M 95 #
— Salida correcta: 8342DFW 98.8

Finalidad: Trabajar con la lectura anticipada y bucles con condicionales dentro. Dificultad Baja.

30. [Aproximación de π por Gregory-Leibniz] 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:

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

Esta es una serie infinita, pues realiza la suma de infinitos términos. Como en Programación 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:

$$\begin{aligned} \frac{\pi}{4} &\approx \sum_{i=0}^{i=\text{tope}} \frac{(-1)^i}{2i+1} = 1 - \frac{1}{2*1+1} + \frac{1}{2*2+1} - \frac{1}{2*3+1} + \dots + \frac{(-1)^{\text{tope}}}{2*\text{tope}+1} \\ &= 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots + \frac{(-1)^{\text{tope}}}{2*\text{tope}+1} \end{aligned}$$

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

Resuelva este problema de tres formas distintas (no hace falta que entregue tres ejercicios: baste con que incluya las dos primeras soluciones dentro de un comentario):

- a) Use la función `pow` (potencia) de `cmath` para implementar $(-1)^i$
- b) Para cada valor de i , calcule $(-1)^i$ con un bucle, tal y como hizo en el ejercicio de la potencia (problema 23 [Factorial y Potencia])
- c) De una forma más eficiente que las anteriores. Por ejemplo, observe que el valor de $(-1)^i$ es 1 para los valores pares de i y -1 para los impares

Recuerde que, para visualizar 15 cifras decimales, por ejemplo, debe incluir la sentencia `cout.precision(15)`; antes de realizar la salida en pantalla.

Ejemplo de entrada: 1000 — Salida correcta: 3.14259165433954

Ejemplo de entrada: 100000 — Salida correcta: 3.14160265348972

Ampliación:

Es fácil demostrar que el valor absoluto de cada sumando se va haciendo cada vez más pequeño, por lo que cuantos más sumandos incluyamos, mejor será la aproximación. En cualquier caso, la serie *converge* al valor correcto de forma muy lenta. Habría que incluir cinco mil millones de términos para obtener 10 cifras decimales exactas en el cómputo de π .

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

31. [Aproximación de π por Wallis] Otra aproximación de π introducida en el siglo XVII por el matemático inglés John Wallis viene dada por:

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

Construya un programa que lea el valor `tope` obligando a que esté entre 1 y cien mil, 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:

- 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`.
- 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

Nota: Dependiendo del orden de ejecución de la multiplicación y la división, es posible que pudiese variar algún decimal.

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

32. [Aproximación de π por Madhava sin usar pow] En el siglo XIV el matemático indio Madhava of Sangamagrama calculó el arco tangente a través de un desarrollo de Taylor (este tipo de desarrollos se ve en la asignatura de Cálculo)

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

Usando como x el valor 1, obtenemos la serie de Leibniz vista en el ejercicio 30:

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

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

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

Por lo tanto, podemos usar la siguiente aproximación:

$$\frac{\pi}{6} \approx \sum_{i=0}^{\text{tope}} \frac{(-1)^i \left(\frac{1}{\sqrt{3}}\right)^{2i+1}}{2i+1}$$

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

Importante: En la implementación de esta solución NO puede usar la función `pow` ni ningún condicional `if`. Se le pide expresamente que para el cómputo de cada término, intente aprovechar los cálculos realizados en la iteración anterior.

Ejemplo de entrada: 1000 — Salida correcta: 3.14159265358979

Ejemplo de entrada: 100000 — Salida correcta: 3.14159265358979

Finalidad: Bucles simples. Aprovechar cálculos de la iteración anterior. Dificultad Media.

33. [Secuencia de temperaturas] 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. Una temperatura es correcta si su valor está en el intervalo $[-90, 60]$ (los extremos representan las temperaturas extremas registradas en la Tierra). La entrada de datos terminará cuando

RELACIÓN DE PROBLEMAS DEL TEMA II. Estructuras de Control

se introduzca una temperatura fuera del rango anterior. El programa 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 18.9 100

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)

Considere los siguientes consejos:

- Tendrá que leer sobre dos variables `anterior` y `actual`, para así poder comparar el valor actual con el anterior.
- Se recomienda que use la técnica de lectura anticipada, por lo que tendrá que leer un primer valor y comprobar si está en el rango adecuado:

```
cin >> anterior;
final_entrada_datos = anterior < MIN_TEMP
                    ||
                    anterior > MAX_TEMP;

while (! final_entrada_datos){
    cin >> actual;
    .....
}
```

Dentro del cuerpo del bucle tendrá que comparar el valor `actual` con los extremos del rango de temperaturas, tal y como se hizo antes de entrar al bucle con el valor `anterior`. Esto hace que repitamos un código muy parecido. Lo resolveremos cuando veamos las funciones.

Ejemplo de entrada: 17.2 17.3 16.2 16.4 17.1 19.2 18.9 100

-- Salida correcta: Inicio: 3 Longitud: 4

Ejemplo de entrada: 17.2 17.3 16.2 16.4 17.1 19.2 100

-- Salida correcta: Inicio: 3 Longitud: 4

Ejemplo de entrada: 17.2 17.3 100

-- Salida correcta: Inicio: 1 Longitud: 2

Ejemplo de entrada: 17.2 15.3 100

-- Salida correcta: Inicio: 1 Longitud: 1

Ejemplo de entrada: 17.2 100

-- Salida correcta: Inicio: 1 Longitud: 1

Ejemplo de entrada: 100

-- Salida correcta: Inicio: 1 Longitud: 0

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

RELACIÓN DE PROBLEMAS DEL TEMA II. Estructuras de Control

34. [Factorial y Potencia: bucle `for`] Retome la solución del ejercicio 23 [Factorial y Potencia] y modifíquelo para que el bucle principal sea un bucle `for` en vez de un bucle `while`.

Finalidad: Ejercitar los bucles `for`. Dificultad Baja.

35. [Intervalo: bucle `for`] Retome la solución del ejercicio 20 [Intervalo: Lectura de valores] y modifíquelo para que el bucle principal sea un bucle `for` en vez de un bucle `while`.

Finalidad: Ejercitar los bucles `for`. Dificultad Baja.

36. [Pirámide] Cree un programa que ofrezca en pantalla la siguiente salida:

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

Modifique la solución para que se lea desde teclado el valor inicial y el número de filas a imprimir. En el ejemplo anterior el valor inicial era 1 y se imprimía un total de 6 filas.

Finalidad: Ejercitar los bucles anidados. Dificultad Baja.

37. [Cuadrado] Cree un programa que ofrezca en pantalla la siguiente salida:

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

Modifique la solución para que se lea desde teclado el valor inicial y el número de filas a imprimir. En el ejemplo anterior el valor inicial era 1 y se imprimía un total de 6 filas.

Finalidad: Ejercitar los bucles anidados. Dificultad Media.

38. [Divisores anidado] Retome la solución del ejercicio 19 [Divisores de un entero] de esta Relación de Problemas. Modifíquelo para que el programa principal lea un número entero positivo `tope` e imprima en pantalla los divisores de todos y cada uno de los números positivos menor o iguales que `tope`

Ejemplo de entrada: 6

— — Salida correcta:

Divisores de 6: 2 3
Divisores de 5: Ninguno
Divisores de 4: 2
Divisores de 3: Ninguno
Divisores de 2: Ninguno

Finalidad: Ejercitar los bucles anidados. Dificultad Baja.

39. [Interés bancario anidado] Sobre la solución del ejercicio 24 [Interés bancario (capital reinvertido)] de esta relación de problemas, se pide lo siguiente. Supondremos que sólo pueden introducirse intereses enteros (1, 2, 3, etc). Se pide calcular el capital obtenido al término de cada año, pero realizando los cálculos para todos los tipos de interés enteros menores o iguales que el introducido (en pasos de 1). Por ejemplo, si el usuario introduce un interés igual a 5 y un número de años igual a 3, hay que mostrar el capital ganado al término de cada uno de los tres años a un interés del 1 %, a continuación, lo mismo para un interés del 2 % y así sucesivamente hasta llegar al 5 %. El programa debe mostrar una salida del tipo:

Cálculos realizados al 1%:

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

Cálculos realizados al 2%:

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

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

40. [Muy divisible] ([Examen Prácticas Noviembre 2019](#)) Diremos que un número es *muy divisible* si, dado un entero k , el número tiene k o más divisores propios (no se consideran ni el 1 ni el propio valor).

Construya un programa que lea un entero cualquiera \min desde teclado. A continuación debe leer otro entero \max obligando, con un filtro de entrada de datos, a que sea mayor o igual que \min . Finalmente el programa leerá un valor entero k con otro filtro para que sea mayor o igual que 1.

El programa imprimirá por pantalla aquellos números en el rango $[\min, \max]$ que tienen k o más divisores (es decir, los números *muy divisibles* dado el valor de k).

Por ejemplo, si $\min = 78$, $\max = 90$ y $k = 3$, el programa debe imprimir 78 80 81 84 88 90 ya que esos son los números en el rango $[78, 90]$ que tienen 3 o más divisores.

Ejemplo de entrada: 78 90 3 — Salida correcta: 78 80 81 84 88 90

Finalidad: Ejercitar los bucles anidados. Dificultad Media.

41. [Tarifa aérea: múltiples billetes] Recupere la solución del ejercicio 26 [Tarifa aérea con filtro de entrada de datos] disponible en:

https://decsai.ugr.es/jccubero/FP/II_TarifaAereaFiltroEntrada.cpp

Modifique dicha solución para que el programa calcule la tarifa final de una serie de billetes. Para ello, cada vez que se introduzcan los datos de un nuevo billete, el usuario introducirá el carácter 'N' (Nuevo). La entrada de datos finaliza con el carácter '#'. Si se introduce un carácter distinto, el programa pedirá un nuevo carácter.

Recuerde que el primer dato a leer era el número de kilómetros del trayecto y el segundo dato corresponde al número de puntos del cliente. En el ejercicio 26 [Tarifa aérea con filtro de entrada de datos] ya se había programado el filtro de entrada para los datos de los kilómetros y número de puntos.

En el siguiente ejemplo, observe que el nuevo filtro de entrada de datos *rechaza* los caracteres J K ya que no son ni N ni #.

Ejemplo de entrada:

J K N -2 -5 200 1300 -450 250 P N 650 1300 -450 90 #

— Salida correcta: 144 185

Finalidad: Filtros de entrada de datos. Bucles anidados. Lectura de datos con terminador. Dificultad Baja.

42. [Mayor nota media] (*Examen Prácticas Noviembre 2019*)

Se quiere calcular la máxima nota media de evaluación continua de un conjunto de alumnos. Para ello, se anota en un fichero un número entero con el código del alumno y las notas que ha conseguido. El número de notas puede variar de un alumno a otro, por lo que se terminará la introducción de las notas con un -1. La entrada de datos finaliza con el código de alumno 0.

Cree un programa que lea las notas desde la entrada por defecto, y calcule el alumno con mayor nota media. Puede suponer que los datos de entrada son siempre correctos.

Por ejemplo, para el siguiente registro de entradas, el alumno con máxima nota es el que tiene identificador 17 con una nota media de 9.5

RELACIÓN DE PROBLEMAS DEL TEMA II. Estructuras de Control

```
11      8 7 6 -1
14      3 -1
 7      9 9 8 7 -1
17      10 9 -1
 8      9 9 -1
15      6 7 5 -1
 5      8 -1
 0
```

El programa imprimirá únicamente los dos enteros (código del mejor alumno y su nota media):

```
17  9.5
```

Finalidad: Bucles anidados. Lectura de datos con terminador. Dificultad Baja.

Ejercicios complementarios de condicionales

43. [Media y desviación típica] Amplíe el ejercicio 19 [Media y desviación] de la relación de problemas I, para que, una vez calculada la media y la desviación, el programa imprima por cada uno de los valores introducidos previamente, si está por encima o por debajo de la media. Por ejemplo:

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

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

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

44. [Coordenadas geográficas (distancia)] Recupere la solución del ejercicio 41 [Coordenadas geográficas (distancia)] de la Relación de Problemas I que puede encontrar en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/I_CoordenadasGeograficas.cpp

Re-escribalo sin utilizar la función `min` (elimine por tanto la inclusión de la biblioteca `algorithm`). Tendrá que calcular el mínimo de dos valores con un condicional simple.

Finalidad: Plantear una estructura condicional sencilla. *Dificultad Baja.*

45. [Codificación de caracteres con algoritmo de rotación. Condicional simple] Reutilice la solución del ejercicio 39 [Codificación de caracteres con algoritmo de rotación] 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.

Una vez obtenida la letra cifrada, descífrela para comprobar que se obtiene la letra de partida.

En este ejercicio puede suponer que la letra introducida será una mayúscula y que el desplazamiento no será mayor que el número de letras mayúsculas.

Ejemplo de entrada: 4 A — Salida correcta: E A

Ejemplo de entrada: 4 Y — Salida correcta: C Y

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

46. [Codificación de caracteres con algoritmo de rotación. Desplazamiento arbitrario] Reutilice la solución del ejercicio 45 [Codificación de caracteres con algoritmo de rotación. Condicional simple] y modifíquela para que tenga en cuenta desplazamientos de cualquier tamaño. Ahora sí puede volver a usar el operador módulo (%)

RELACIÓN DE PROBLEMAS DEL TEMA II. Estructuras de Control

En este ejercicio puede suponer que la letra introducida será una mayúscula.

Ejemplo de entrada: 4 A — Salida correcta: E A

Ejemplo de entrada: 30 A — Salida correcta: E A

Ejemplo de entrada: 56 A — Salida correcta: E A

Ejemplo de entrada: 4 Y — Salida correcta: C Y

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

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

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

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

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

Imprima el salario final en pantalla.

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

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

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

48. Recupere la solución del ejercicio 47 sobre el cómputo de la nómina de los trabajadores de un centro de atención telefónica. Implemente ahora el siguiente criterio para la subida salarial. Se aplicará una subida del 4% a los empleados que han resuelto más de 30 casos y una subida del 2% si el grado de satisfacción media de los usuarios es mayor o igual que 4.0. Ambas subidas son compatibles, es decir, si un trabajador cumple las dos condiciones, se le aplicarán ambas subidas.

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

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

RELACIÓN DE PROBLEMAS DEL TEMA II. Estructuras de Control

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

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

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

Finalidad: Plantear estructuras condicionales consecutivas. Dificultad Baja.

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

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

50. [Tres valores ordenados separando E/S y C con un enumerado] Modifique el ejercicio 49 para que el programa nos diga si los tres valores leídos están ordenados de forma ascendente, ordenados de forma descendente o no están ordenados. Para resolver este problema, debe usar una variable de tipo enumerado.

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

51. Modifique la solución del ejercicio 48 para que ambas subidas salariales sean excluyentes, es decir, si se aplica una, no se aplicará la otra. En el caso de que ambas sean aplicables, debe aplicarse la subida más ventajosa para el trabajador, es decir, la del 4%.

De forma exclusiva:

Más de 30 casos resueltos: +4%

Grado de satisfacción ≥ 4 : +2%

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

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

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

Finalidad: Plantear estructuras condicionales anidadas. Dificultad Baja.

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

Si permanece más de 660 minutos se paga una única tarifa de 31.55 euros

Desde el minuto 0 al 30: 0.0412 euros cada minuto

Desde el minuto 31 al 90: 0.0370 euros cada minuto

Desde el minuto 91 al 120: 0.0311 euros cada minuto

Desde el minuto 121 al 660: 0.0305 euros cada minuto

RELACIÓN DE PROBLEMAS DEL TEMA II. Estructuras de Control

Dado un tiempo de entrada (hora, minuto y segundo) y un tiempo de salida, construya un programa que calcule la tarifa final a cobrar. Para calcular el número de minutos entre los dos instantes de tiempo, puede utilizar la solución del ejercicio 13 [Segundos entre dos instantes] de la Relación de Problemas I.

Ejemplo de entrada: 2 1 30 2 1 29 — Salida correcta: -1

Ejemplo de entrada: 2 1 30 2 1 31 — Salida correcta: 0

Ejemplo de entrada: 2 1 30 2 2 31 — Salida correcta: 0.0412

Ejemplo de entrada: 2 1 30 2 41 31 — Salida correcta: 1.606

Ejemplo de entrada: 2 1 30 3 41 31 — Salida correcta: 3.767

Ejemplo de entrada: 2 1 30 5 41 31 — Salida correcta: 7.439

Ejemplo de entrada: 2 1 30 23 1 1 — Salida correcta: 31.55

Finalidad: Plantear una estructura condicional anidada. Dificultad Baja.

53. Modifique la solución del ejercicio 48 para que también aplique una subida del 3% a los que han resuelto entre 20 y 30 casos:

Entre 20 y 30 casos resueltos: +3%

Más de 30 casos resueltos: +4%

Grado de satisfacción ≥ 4 : +2%

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

Ejemplo de entrada: 7.5 130 24 3 — Salida correcta: 1004.25

Ejemplo de entrada: 7.5 130 24 4 — Salida correcta: 1023.75

Finalidad: Plantear una estructura condicional anidada. Dificultad Baja.

54. 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 en pantalla.

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

Cambiar el programa para incorporar este nuevo criterio.

Finalidad: Plantear una estructura condicional anidada. Dificultad Baja.

55. Modifique la solución del ejercicio 49 (valores ordenados) para que no se mezclen E/S y C (entradas/salidas y cálculos) dentro de la misma estructura condicional.

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

56. Modifique la solución del ejercicio 3 (año bisiesto) para que no se mezclen E/S y C (entradas/salidas y cálculos) dentro de la misma estructura condicional.

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

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

- La renta bruta es la cantidad de dinero íntegra que el trabajador gana.
- La retención fiscal es el tanto por ciento que el gobierno *se queda*.
- La renta neta es la cantidad que le queda al trabajador después de quitarle el porcentaje de retención fiscal, es decir:
$$\text{Renta_neta} = \text{Renta_bruta} - \text{Renta_bruta} * \text{Retención final} / 100$$

Los datos a leer son:

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

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

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

RELACIÓN DE PROBLEMAS DEL TEMA II. Estructuras de Control

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.

Ejercicios complementarios de bucles

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

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

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

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

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

— Salida correcta: 2 1

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

— Salida correcta: 2 1

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

— Salida correcta: Número de intentos sobrepasado

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

60. **[Diferencia entre instantes con filtro de entrada]** Amplíe el ejercicio 13 **[Segundos entre dos instantes]** de la Relación de Problemas I de manera que se obligue al usuario a que el segundo instante introducido sea posterior al primero. Suponemos que ambos instantes corresponden al mismo día. Para comparar los instantes puede usar la solución del ejercicio 14 **[Comparación de dos instantes]** de esta Relación de Problemas.

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

61. **[Números perfectos]** (*Examen Prácticas Noviembre 2019*)

Diremos que un número $n > 1$ es *perfecto* si es igual a la suma de todos sus divisores positivos (incluyendo el 1) excepto él mismo. Por ejemplo, el 6 es perfecto ya que sus divisores son 1, 2 y 3 y se cumple que $6=1+2+3$.

Construya un programa principal que lea un entero cualquiera \min desde teclado obligando a que sea mayor o igual que 1 y otro entero \max obligando a que sea mayor o igual que \min . El programa debe mostrar todos los números perfectos que hay entre \min y \max .

Por ejemplo, si $\min = 13$ y $\max = 500$, el programa debe imprimir 28 496.

Finalidad: Ejercitar los bucles anidados. Dificultad Media.

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

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

En general, si un número es feliz de grado k' también es feliz para cualquier grado $k \geq k'$

Escriba un programa que lea un valor entero n y un valor k . Si el número es feliz para un grado k' menor o igual que k , el programa parará y mostrará dicho grado k' . En caso contrario, el programa nos dirá que no es feliz para cualquier grado menor o igual que k .

Ejemplo de entrada: 13 2

— Salida correcta: Es feliz para cualquier grado ≥ 2

Ejemplo de entrada: 13 5

— Salida correcta: Es feliz para cualquier grado ≥ 2

Ejemplo de entrada: 13 1

— Salida correcta: No es feliz para cualquier grado ≤ 1

Ejemplo de entrada: 203 1

— Salida correcta: No es feliz para cualquier grado ≤ 1

Ejemplo de entrada: 203 7

— Salida correcta: Es feliz para cualquier grado ≥ 3

Ejemplo de entrada: 102 6

— Salida correcta: No es feliz para cualquier grado ≤ 6

Finalidad: Ejercitar los bucles anidados. Dificultad Media.

63. **[Gaussiana con un menú]** Recupere la solución del ejercicio 17 (función gaussiana) de la relación de problemas I. Se pide crear un menú principal para que el usuario pueda elegir las siguientes opciones:

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

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

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

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

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

— Salida correcta:

```
f(11)=0.0782085
f(11.5)=0.0793905
f(12)=0.0797885
f(12.5)=0.0793905
f(13)=0.0782085
```

Finalidad: Ejercitar los bucles anidados. Dificultad Media.

64. **[Bits to char]** (*Examen Julio 2017*) 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 '0'. El programa debe ir construyendo el entero 79 como suma de potencias de 2 ($0 * 2^7 + 1 * 2^6 + \dots$)

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 → 0, 01101011 → k, y 00100001 → !. 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.

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

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

Finalidad: Ejercitar el uso de bucles. Dificultad Baja.

66. Amplie el ejercicio 37 (Población) de la relación de problemas I.

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

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

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

Por ejemplo, para la siguiente entrada

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

el programa debe devolver lo siguiente:

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

Ejemplo de entrada: 1375570814 32 12 7 3
— Salida correcta: 1490027497 27 2824131580

Finalidad: Ejemplo básico de asignación acumulada. Dificultad Baja.

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

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

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

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

68. Se quiere construir un programa para leer los datos necesarios del ejercicio 53 de la subida salarial.

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

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

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

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

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

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

— Salida correcta: 3 2 2.5

Ejemplo de entrada: 7.5 -1

— Salida correcta: No se introdujo ningún caso

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

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

Finalidad: Ejercitar los bucles con condiciones de salida complejas. Dificultad Media.

70. **[RLE]** 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

RELACIÓN DE PROBLEMAS DEL TEMA II. Estructuras de Control

es un mecanismo de compresión de datos (zip) sin pérdidas. Se aplica, por ejemplo, para comprimir los ficheros de imágenes en las que hay zonas con los mismos datos (fondo blanco, por ejemplo). Realice un programa que lea una secuencia de números naturales terminada con un número negativo y la codifique mediante el método RLE.

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

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

71. [Pinta dígitos arbitrario] En el ejercicio 36 [Pinta dígitos] de la Relación de Problemas I se pedía escribir un programa que leyese un valor entero de tres dígitos e imprimiese los dígitos separados por un espacio en blanco. Haga lo mismo pero para un número entero arbitrario. Por ejemplo, si el número es 3519, la salida sería:

3 5 1 9

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

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

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

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

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

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

Dificultad Media.

73. Amplíe el ejercicio 3 (año bisiesto). El programa pedirá los valores de dos años obligando a que estén entre el año cero y 2100. A continuación, el programa mostrará todos los años bisiestos comprendidos entre los años anteriores.

RELACIÓN DE PROBLEMAS DEL TEMA II. Estructuras de Control

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

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

- a) Modifique la solución del ejercicio 19 de forma que el filtro de entrada usado para leer la variable `tope`, se haga con un bucle pre-test `while`.
- b) Modifique la solución del ejercicio 24 sustituyendo el bucle `while` por un `do while`. Observad que debemos considerar el caso en el que el número de años leído fuese cero.

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

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

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

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

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

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

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

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

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

es decir:

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

Ejemplo de entrada: 3 — Salida correcta: 9

Ejemplo de entrada: 4 — Salida correcta: 33

Ejemplo de entrada: 6 — Salida correcta: 873

Finalidad: Ejercitar los bucles anidados. Dificultad Media.

77. Resuelva el ejercicio 76 sin utilizar bucles anidados, es decir, debe usar un único bucle.
Finalidad: Aprovechar en una iteración los cálculos hechos en la iteración anterior.
Dificultad Media.

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

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

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

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

Finalidad: Ejercitar los bucles anidados. Dificultad Media.

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

Finalidad: Ejercitar los bucles anidados. Dificultad Baja.

80. Supongamos una serie numérica cuyo término general es:

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

Es decir, la serie la forman los siguientes términos:

$$a_1 = a_1$$

$$a_2 = a_1 r$$

$$a_3 = a_1 r^2$$

$$a_4 = a_1 r^3$$

...

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

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

Se proponen dos alternativas:

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

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

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

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

¿Qué solución es preferible en términos de eficiencia?

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

81. Reescribid la solución a los ejercicios 19 (divisores) y 24 (interés) usando un bucle `for`

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

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

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

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

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

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

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

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

```
#include <iostream>
#include <ctime>
```

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

int main(){
    const int MIN = 1, MAX = 100;
    const NUM_VALORES = MAX-MIN + 1;           // rango
    int incognita;                             // número generado
    time_t tiempo;

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

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

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

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

Dificultad Media.

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

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

Finalidad: Ejercitar los bucles anidados. Dificultad Baja.

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

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

Finalidad: Ejercitar los bucles anidados. Dificultad Media.

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

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

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

Dificultad Baja.

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

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

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

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

Dificultad Baja.

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

Dificultad Baja.

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

Dificultad Baja.

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

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

para un valor n dado.

Dificultad Baja.

RELACIÓN DE PROBLEMAS DEL TEMA II. Estructuras de Control

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

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

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

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

Dificultad Baja.

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

Por ejemplo, con la entrada

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

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

Dificultad Baja.

93. 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.

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

Dificultad Baja.

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

Dificultad Baja.

RELACIÓN DE PROBLEMAS DEL TEMA II. Estructuras de Control

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

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

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

En este caso, sí se encuentra.

Dificultad Media.

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

Dificultad Media.

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

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

Dificultad Media.

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

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

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

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

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

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

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

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

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

Se pueden construir aproximaciones al número áureo mediante la fórmula $a_n = \frac{f_{n+1}}{f_n}$ siendo f_n el número de Fibonacci de orden n (ver problema 99).

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

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

La entrada del programa será el valor de δ y la salida el valor de n . Por ejemplo, para un valor de $\delta = 0.1$ el valor de salida es $n = 4$

Dificultad Media.

101. Una *sucesión alícuota* es una sucesión iterativa en la que cada término es la suma de los divisores propios del término anterior. La sucesión alícuota que comienza con el

entero positivo k puede ser definida formalmente mediante la función divisor σ_1 de la siguiente manera:

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

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

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

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

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

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

102. [Cuadro de límites de velocidad] La Dirección General de Tráfico publica en su página web la sanción a aplicar cuando se sobrepasa el límite de velocidad establecido en una vía. La sanción consta de una multa económica y una detracción del número de puntos del carnet del infractor.

El siguiente cuadro está publicado en

https://sede.dgt.gob.es/Galerias/tramites-y-multas/alguna-multa/consulta-de-sanciones-por-exceso-velocidad/cuadro_velocidad.pdf

RELACIÓN DE PROBLEMAS DEL TEMA II. Estructuras de Control



Cuadro para excesos de velocidad (Fecha de entrada en vigor 25/05/2010)

Limite		30	40	50	60	70	80	90	100	110	120	Multa	Puntos
Exceso de velocidad	Grave	31 50	41 60	51 70	61 90	71 100	81 110	91 120	101 130	111 140	121 150	100	-
		51 60	61 70	71 80	91 110	101 120	111 130	121 140	131 150	141 160	151 170	300	2
		61 70	71 80	81 90	111 120	121 130	131 140	141 150	151 160	161 170	171 180	400	4
		71 80	81 90	91 100	121 130	131 140	141 150	151 160	161 170	171 180	181 190	500	6
	Muy Grave	81	91	101	131	141	151	161	171	181	191	600	6

Escriba un programa en C++ para imprimir los límites inferiores indicados dentro del cuadro. Es decir, el programa debe imprimir lo siguiente:

```
31 41 51 61 71 81 91 101 111 121
51 61 71 91 101 111 121 131 141 151
61 71 81 111 121 131 141 151 161 171
71 81 91 121 131 141 151 161 171 181
81 91 101 131 141 151 161 171 181 191
```

Debe usar un bucle anidado en otro y establecer los incrementos convenientes. Por ejemplo, el incremento entre dos columnas consecutivas de una misma fila es de 10, salvo el incremento que hay entre la tercera y cuarta columnas que es de 10 para la primera fila, de 20 para la segunda y de 30 para la tercera fila y siguientes.

Finalidad: Ejercitar los bucles anidados. Dificultad Media.

RELACIÓN DE PROBLEMAS DEL TEMA III. Vectores y Matrices

En todos los ejercicios, tenga en cuenta lo siguiente:

- Siempre ha de diseñar una batería de pruebas, que intente garantizar que la solución propuesta funcione correctamente en cualquier situación extrema.
- Para leer datos de tipo `char` tenemos que hacerlo de una forma especial. Como ya vimos en las transparencias del primer tema, la lectura de datos se realiza a través de un *buffer* intermedio. Cuando leemos un dato `caracter` de tipo `char` en la forma `cin >> caracter;`, la lectura se salta todos los espacios en blanco (' '), tabuladores ('\t') y retornos de carro ('\n') previos que hubiese en el buffer. Si queremos leer estos caracteres especiales (en especial nos interesa leer el espacio en blanco ' ') tenemos que hacerlo con `caracter = cin.get()`.

Cada vez que se ejecute `cin.get()` el programa lee un carácter del buffer. Dicho carácter puede ser un carácter *habitual* como por ejemplo 'a', pero también puede ser un espacio en blanco, un tabulador o un retorno de carro.

Para ver con más detalle cómo funciona, supongamos que ejecutamos el siguiente código:

```
char vector[100];
...
ultimo = 0;
caracter = cin.get();

while (caracter != '#'){
    vector[ultimo] = caracter;
    ultimo++;
    caracter = cin.get();
}
```

Supongamos que el buffer está vacío: por lo tanto, al ejecutarse el primer `cin.get()` se piden datos al teclado. Cada vez que se pulse la tecla <ENTER> los caracteres introducidos desde el teclado pasarán al buffer, incluido el propio carácter '\n' correspondiente al <ENTER>. Así pues, si desde el teclado introducimos lo siguiente:

Teclado -> a b<TABULADOR>#<ENTER>

en el buffer tendremos los siguientes datos:

Buffer -> a b\t#\n

Por lo tanto, el vector quedaría de la siguiente forma:

```
vector = ['a', ' ', 'b', '\t']
```

Observe que el buffer aún no estaría vacío sino que tendría lo siguiente:

```
Buffer -> \n
```

por lo que si ahora se ejecutase otro `cin.get()` leería dicho carácter `'\n'`.

Una situación similar se produce en el siguiente ejemplo. Supongamos que leemos un entero, por ejemplo, con `cin >> entero;` y desde el teclado introduce 75 y pulsa la tecla ENTER (`'\n'`); en el buffer se almacena lo siguiente:

```
Teclado -> 75<ENTER>
```

```
Buffer -> 75\n
```

el valor 75 desaparece del buffer y se le asigna al dato `entero`, pero el carácter `'\n'` sigue en el buffer, por lo que si ahora ejecuta `character = cin.get()` a la variable `character` se le asignará `'\n'`.

Para evitar estas situaciones, a lo largo de los ejercicios de esta relación de problemas, fomentaremos la lectura de una serie de caracteres *de una sola vez*. Por ejemplo, si vamos a leer dos secuencias de caracteres hasta llegar a un terminador `'#'` introduciremos los datos desde el teclado todos seguidos y pulsaremos `<ENTER>` sólo al final (y no después de cada terminador `'#'`):

```
:-)
```

```
Teclado -> Una secuencia#Otra Secuencia#<ENTER>
```

```
Buffer -> Una secuencia#Otra Secuencia#\n
```

```
:- (
```

```
Teclado -> Una secuencia#<ENTER>Otra Secuencia#<ENTER>
```

```
Buffer -> Una secuencia#\nOtra Secuencia#\n
```

- Algunos ejercicios usan un vector muy grande. Para que el vector con dicho texto quepa en la pila, debe hacer los siguientes cambios:

- Declare una constante de tamaño del vector que permita almacenar todos los datos. Por ejemplo, para almacenar 20 millones de datos de un tipo de dato cualquiera, definiríamos el siguiente vector:

```
const int TAMANIO = 20000000;  
<tipo de dato> vector[TAMANIO];
```

Si se prefiere, para que quede más claro el número de datos reservados en memoria, puede usar una constante `double` en formato científico (`2e+7`) y dejar que C++ haga el casting a `int` automáticamente:

```
const int TAMANIO = 2e+7; // int = double  
<tipo de dato> vector[TAMANIO];
```

RELACIÓN DE PROBLEMAS DEL TEMA III. Vectores y Matrices

- Aumente el tamaño de la pila asignada por el sistema operativo al programa generado por el compilador. Para ello, debe seleccionar desde DevC++:

Herramientas -> Opciones del Compilador ->

Señale la opción

Añadir los siguientes comandos al llamar al compilador y en la caja de texto introduzca lo siguiente (todo seguido sin espacios en blanco):

`-Wl,--stack,n`

donde `n` será un número entero que indicará el número de bytes que va a permitir almacenar en la pila. Por ejemplo, si necesita un vector de $2e+7$ datos de tipo `int`, necesitará un mínimo de $2e+7 * 4$ bytes, es decir $8e+7$ bytes. Reserve algo más si va a utilizar varios vectores del tamaño anterior. Por lo tanto, un valor de `n` suficiente en este caso podría ser $2e+8$. Así pues, pondríamos lo siguiente:

`-Wl,--stack,200000000`

Tenga en cuenta que aquí no puede usar la notación científica.

Problemas sobre vectores

1. [Divisores] Recupere la solución del ejercicio 19 [Divisores de un entero] de la Relación de Problemas II que puede encontrar en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/II_Divisores.cpp

Modifíquelo para separar los cálculos de las entradas y salidas de datos. Para ello, se pide que cada vez que encuentre un divisor lo guarde en un vector `divisores`. Una vez construido el vector, en un bucle aparte, debe imprimir sus componentes en pantalla.

Ejemplo de entrada: 16 — Salida correcta: 2, 4, 8

2. [Palíndromo e Invierte] Construya un programa que declare un vector de caracteres de tamaño 100. Lea caracteres desde la entrada por defecto hasta que se introduzca el carácter # y almacénelos en el vector anterior (el carácter terminador # no se añade al vector). En la lectura de los caracteres, debe tener en cuenta que no se deben introducir más caracteres que el tamaño del vector (100 en este ejemplo)

Para leer los caracteres (incluido el espacio en blanco), haga uso de la instrucción `character = cin.get()` (en vez de `cin >> character`) tal y como se indica en la página Problemas-67.

Implemente algoritmos para realizar las siguientes tareas:

- a) Comprobar si el vector es un palíndromo, es decir, que se lee igual de izquierda a derecha que de derecha a izquierda. Por ejemplo, {'a', 'b', 'b', 'a'} sería un palíndromo, pero {'a', 'c', 'b', 'a'} no lo sería. Si la secuencia tiene un número impar de componentes, la que ocupa la posición central es descartada, por lo que {'a', 'b', 'j', 'b', 'a'} sería un palíndromo.
- b) Invertir el vector. Si éste contenía, por ejemplo, los caracteres {'m', 'u', 'n', 'd', 'o'}, después de llamar al método se quedará con {'o', 'd', 'n', 'u', 'm'}.

Una vez leídos los caracteres y almacenados en el vector, el programa debe determinar si es un palíndromo. En caso negativo, debe invertirlo (colocar sus componentes en orden inverso) y mostrar el resultado en pantalla.

Ejemplo de entrada: a# — Salida correcta: Es un palíndromo

Ejemplo de entrada: abcba# — Salida correcta: Es un palíndromo

Ejemplo de entrada: abccba# — Salida correcta: Es un palíndromo

Ejemplo de entrada: abcdab#

— Salida correcta: No es un palíndromo. Secuencia invertida: abdcba

Finalidad: Recorrer las componentes de un vector. Dificultad Baja.

3. [Muy divisible con vectores] Recupere la solución del ejercicio 40 [Muy divisible] que se encuentra en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/II_MuyDivisible.cpp

Modifique el código para separar Entradas/Salidas de Cómputos. Para ello, tiene que almacenar todos los valores que sean muy divisibles en un vector. Un vez haya terminado de almacenar todos los números muy divisibles, procederá a imprimirlos en pantalla en un bloque de código aparte.

Ejemplo de entrada: 78 90 3 — Salida correcta: 78 80 81 84 88 90

Finalidad: Empezar a trabajar con vectores. Dificultad Baja.

4. [Sustituir carácter por vector (con vector auxiliar)] Dado un vector de caracteres, queremos sustituir todas las apariciones de un carácter y poner en su lugar el contenido de otro vector.

Por ejemplo, si tenemos el vector [u n o a d o s a a], el resultado de sustituir las apariciones del carácter 'a' por el nuevo vector [T T U] sería [u n o T T U d o s T T U T T U]

Resolveremos este problema de varias formas a lo largo de esta Relación de Problemas. En este ejercicio, se construirá un tercer vector sustituido con el resultado pedido.

Construya un programa que lea caracteres hasta que se introduzca # lo que formará el primer vector (v). A continuación lea el carácter a_borrar que se va a eliminar de v. Finalmente, el programa leerá los caracteres que formarán el vector a_insertar que sustituirán cada aparición de a_borrar. El terminador de entrada de caracteres para el vector a_insertar es también el carácter #

Para leer los caracteres (incluido el espacio en blanco), haga uso de la instrucción `caracter = cin.get()` (en vez de `cin >> caracter`) tal y como se indica en la página Problemas-67.

El programa construirá e imprimirá en pantalla un tercer vector sustituido que contendrá los caracteres de v pero reemplazando todas las apariciones de a_borrar por los caracteres del vector a_insertar. Si el vector a_insertar contuviese el carácter a_borrar, dichas apariciones no se eliminan, tal y como puede apreciarse en el último ejemplo que aparece al final de este enunciado.

Para realizar la tarea pedida, se recomienda que implemente el siguiente algoritmo:

```
Recorrer las componentes -i- del vector v
  Si v[i] == a_borrar
    Añadir a sustituido todas las componentes
    del vector a_insertar
```

si no

Añadir a sustituido la componente $v[i]$

Puede utilizar el esbozo del programa disponible en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/III_SustituyeCaracterVectorEsbozo.cpp

Ejemplo de entrada: u n o a d o s a a # T T U # a

— Salida correcta: u n o T T U d o s T T U T T U

Ejemplo de entrada: u n o a d o s a a # T a U # a

— Salida correcta: u n o T a U d o s T a U T a U

Finalidad: Trabajar con vectores auxiliares. Dificultad Baja.

5. **[Sumatoria]** (*Examen Marzo 2019*) Dado un vector de enteros v y dado un número T , se quiere ver si hay una serie consecutiva de elementos del vector que sume T . Por ejemplo, si $v = [4, 1, 3, 9, 2, -20]$:

- Si $T = 6$, no hay ninguna secuencia.
- Si $T = 12$, sí hay. Sería la secuencia $[3, 9]$
- Si $T = 14$, sí hay. Sería la secuencia $[3, 9, 2]$

Construya un programa que lea los datos en el siguiente orden:

- a) En primer lugar, el número n de elementos que se van a introducir
- b) En segundo lugar, los n enteros (puede haber positivos y negativos)
- c) Finalmente, el valor T

El programa debe imprimir en pantalla el índice en el que comienza la secuencia.

Ejemplo de entrada: 5 4 1 3 9 2 6

— Salida correcta: No hay ninguna secuencia

Ejemplo de entrada: 5 4 1 3 9 2 12

— Salida correcta: Secuencia encontrada a partir de la posición 2

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

6. **[Frecuencias caracteres]** En este ejercicio vamos a contar el número de ocurrencias de ciertos caracteres en un texto.

El programa leerá una serie de caracteres hasta llegar al terminador @. Estos caracteres se introducirán en un vector llamado `a_buscar`. A continuación, el programa leerá otro conjunto de caracteres hasta llegar al terminador @ y se almacenarán en un vector llamado `texto`. Supondremos que los caracteres introducidos son de la tabla ASCII (no se introducirán vocales acentuadas, la ñ, etc)

Para leer los caracteres (incluido el espacio en blanco), haga uso de la instrucción `caracter = cin.get()` (en vez de `cin >> caracter`) tal y como se indica en la página [Problemas-67](#).

Una vez leídos los dos vectores, el programa debe calcular la frecuencia (el número de ocurrencias) de cada uno de los caracteres del vector `a_buscar` dentro del vector `texto` y almacenar dichos conteos en un vector `frecuencias`. Declare el vector `frecuencias` con un máximo de 256 componentes (tantas como caracteres hay en la tabla ASCII). Finalmente el programa imprimirá en pantalla los conteos correspondientes.

Si lo desea, puede usar el esbozo del programa que puede encontrar en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/III_FrecuenciasEsbozo.cpp

Se le pide que resuelva este problema de dos formas distintas:

- El vector `frecuencias` contendrá las frecuencias de todos los caracteres de `texto` (sin tener en cuenta que estén o no en el vector `a_buscar`).
En este caso, la componente `i` del vector `frecuencias` contendrá la frecuencia del carácter que tiene como orden `i` en la tabla ASCII. Por ejemplo, `frecuencias[66]` contendrá el número de ocurrencias del carácter 'B' dentro de `texto` (da igual si dicho carácter está o no en el vector `a_buscar`).
- El vector `frecuencias` contendrá sólo las frecuencias de los caracteres de `texto` que están en el vector `a_buscar`.
En este caso, la componente `i` del vector `frecuencias` contendrá la frecuencia del carácter `a_buscar[i]` (el número de ocurrencias dentro del vector `texto`). Por ejemplo, si el vector `a_buscar` contiene los caracteres ['a' , 'J'], entonces `frecuencias[0]` contendrá la frecuencia de 'a' y `frecuencias[1]` contendrá la frecuencia de 'J'.

Ejemplo de entrada: `JasP@Juan Carlos Cubero@`

— Salida correcta:

`J: 1 a: 2 s: 1 P: 0`

Observe que el orden de salida de las frecuencias coincide con el orden en el que se introdujeron los caracteres del vector `a_buscar`.

Finalidad: Recorrer varios vectores y crear un índice para un acceso directo a la componente. Dificultad Baja.

7. **[Comprobación Fecha]** Se quiere construir un programa que compruebe si una fecha es correcta. El programa pedirá los datos de día mes y año e indicará si la fecha correspondiente es correcta o no. El programa pedirá los datos de varias fechas, hasta que el usuario introduzca cualquier valor negativo en el dato del día.

Para resolver este problema, debe tener en cuenta lo visto sobre los años bisiestos en el ejercicio 27 [Expresiones lógicas] de la Relación de Problemas I así como los días que trae cada mes (Noviembre 30, Diciembre 31, etc) Se aconseja usar un vector con tantas componentes como meses hay (12).

Separe la parte de cálculos con las Entradas/Salidas del programa.

Ejemplo de entrada: 10 12 1967 10 15 1967 10 1 2015 -1

-- Salida correcta: SI NO SI

Finalidad: Usar vectores auxiliares para realizar recorridos de datos. Dificultad Baja.

8. [Cuenta Mayúsculas] Construya un programa que vaya leyendo caracteres hasta que se encuentre un punto '.' y cuente el número de veces que aparece cada una de las letras mayúsculas. Imprima el resultado.

Una posibilidad sería declarar un vector `contador_mayusculas` con tantas componentes como letras mayúsculas hay ('Z' - 'A' + 1) y conforme se va leyendo cada carácter, ejecutar lo siguiente:

```
cin >> letra;
```

```
if (letra == 'A')
    contador_mayusculas[0] = contador_mayusculas[0] + 1;
else if (letra == 'B')
    contador_mayusculas[1] = contador_mayusculas[1] + 1;
else if (letra == 'C')
    contador_mayusculas[2] = contador_mayusculas[2] + 1;
else ....
```

Sin embargo, este código es muy redundante. Como solución se propone calcular de forma directa el índice entero que le corresponde a cada mayúscula, de forma que todos los if-else anteriores los podamos resumir en una **única** sentencia del tipo:

```
contador_mayusculas[indice] = contador_mayusculas[indice] + 1;
```

Finalidad: Acceder a las componentes de un vector con unos índices que representen algo. Dificultad Baja.

9. [Mayor desnivel] Supongamos un vector de enteros relativo a un conjunto de alturas de un track GPS. Cada entero representa la altura de la posición GPS en un instante dado. Se quiere calcular lo siguiente:
- Máximo desnivel (máxima diferencia en valor absoluto) entre dos alturas consecutivas.

Por ejemplo, si el vector es 1 3 2 4 7 5, las diferencias son +2 -1 +2 +3 -2, siendo 3 la máxima en valor absoluto. Si el vector fuese 1 3 2 4 1 2, las diferencias serían +2 -1 +2 -3 +1, siendo 3 la máxima en valor absoluto.

- Desnivel acumulado positivo.

La idea es ir buscando puntos consecutivos en los que la altura vaya aumentando. El desnivel acumulado se calcula como la suma de todas esas diferencias positivas.

Por ejemplo, si el vector es 1 3 2 4 7 5, las diferencias son +2 -1 +2 +3 -2, por lo que el desnivel acumulado positivo será 7 (la suma de +2 +2 +3)

Construir un programa que, en primer lugar, lea un número entero que indique el número de valores de altura que se van a introducir. A continuación el programa leerá dichos valores y los almacenará en un vector. El programa calculará el máximo desnivel y el desnivel acumulado positivo y los imprimirá en pantalla.

Ejemplo de entrada: 6 1 3 2 4 7 5

— Salida correcta: 3 7

Ejemplo de entrada: 2 4 1

— Salida correcta: 3 0

Ejemplo de entrada: 1 7

— Salida correcta: No hay datos suficientes

Finalidad: Recorridos sobre un vector. Dificultad Baja.

10. **[Moda]** Se quiere calcular la moda de un vector de caracteres, es decir, el carácter que más veces se repite. Por ejemplo, si el vector fuese

{ 'l', 'o', 's', ' ', 'd', 'o', 's', ' ', 'c', 'o', 'f', 'r', 'e', 's' }

los caracteres que más se repiten son 'o' y 's' con un total de 3 apariciones. La moda sería cualquiera de ellos, por ejemplo, el primero encontrado 'o'.

Construya un programa que lea caracteres hasta que el usuario introduzca el carácter #. Almacene todos los valores en un vector de caracteres llamado `texto`. A continuación, calcule la moda y muéstrela en pantalla junto con su frecuencia (el número de apariciones).

Para leer los caracteres (incluido el espacio en blanco), haga uso de la instrucción `caracter = cin.get()` (en vez de `cin >> caracter`) tal y como se indica en la página [Problemas-67](#).

Resuelva este problema aplicando el siguiente algoritmo que utiliza un vector auxiliar procesados para almacenar aquellos caracteres cuya frecuencia ya se ha contado:

```
Recorrer -i- el vector texto
    actual = texto[i]
```

Si actual no está en el vector procesados:
Añadir actual a procesados
Contar el número de ocurrencias de actual
en el vector texto -a partir de la posición i+1-

Ejemplo de entrada: JuanCarlosCubero#

— Salida correcta: u 2

Finalidad: Recorridos sobre un vector. Dificultad Media.

11. **[Moda (versión 2)]** Se quiere calcular la moda de un vector de caracteres tal y como se ha enunciado en el ejercicio 10 utilizando ahora un vector auxiliar conteos para almacenar los conteos de todos los caracteres. Podemos suponer que el texto sólo contiene caracteres del alfabeto inglés, por lo que podemos usar un vector conteos de 256 componentes, de forma que `conteos[i]` contiene el número de veces que el carácter con orden *i* en la tabla ASCII está en el texto.

Por ejemplo, si el texto es AbcAAb, el vector conteos tendrá todas sus componentes a cero excepto `conteos[65] = 3`, `conteos[98] = 2`, `conteos[99] = 1`.

Una vez haya almacenado los conteos, recorra el vector para calcular el máximo.

Ejemplo de entrada: JuanCarlosCubero#

— Salida correcta: u 2

Finalidad: Recorridos sobre un vector. Dificultad Baja.

12. **[Sistema de D'Hondt] (Examen Febrero 2017)** El sistema D'Hondt es el método que se utiliza en España para asignar los escaños del Congreso de los Diputados. Se quiere construir un programa que lea el número total de escaños a distribuir, el número de partidos que han participado en las elecciones y los votos obtenidos por cada uno de ellos. El programa mostrará cuántos escaños obtuvo cada partido.

La asignación de los escaños se hace a través de un proceso iterativo en el que en cada iteración se asigna un escaño a un partido y así hasta llegar al número total de escaños a repartir. En una determinada iteración, un partido se llevará un escaño si tiene el mayor cociente de D'Hondt, definido éste como sigue:

$$\text{Cociente de D'Hondt} = V_i / (S_i + 1)$$

dónde V_i es el número total de votos obtenidos en las elecciones por el partido *i* y S_i es el número de escaños asignados hasta esa iteración al partido *i*.

Abajo se muestra un ejemplo de este proceso iterativo con 5 escaños a repartir y 4 partidos. Un asterisco (*) nos indica que el partido correspondiente (la columna) se lleva el escaño a asignar en esa iteración (la fila).

Ejemplo de entrada: 5 4 340000 280000 160000 60000

— Salida correcta: 2 2 1 0

RELACIÓN DE PROBLEMAS DEL TEMA III. Vectores y Matrices

Votos	Partido A $V_A = 340000$	Partido B $V_B = 280000$	Partido C $V_C = 160000$	Partido D $V_D = 60000$
Escaño 1	(340000/1 =) 340000*	(280000/1 =) 280000	(160000/1 =) 160000	(60000/1 =) 60000
Escaño 2	(340000/2 =) 170000	(280000/1 =) 280000*	(160000/1 =) 160000	(60000/1 =) 60000
Escaño 3	(340000/2 =) 170000*	(280000/2 =) 140000	(160000/1 =) 160000	(60000/1 =) 60000
Escaño 4	(340000/3 =) 113333	(280000/2 =) 140000	(160000/1 =) 160000*	(60000/1 =) 60000
Escaño 5	(340000/3 =) 113333	(280000/2 =) 140000*	(160000/2 =) 80000	(60000/1 =) 60000
E. asignados	2	2	1	0

Ejemplo de entrada: 7 5 340000 280000 160000 60000 15000

— Salida correcta: 3 3 1 0 0

Nota:

La ley electoral vigente en España fija de antemano un número de escaños en cada provincia. El método de reparto de escaños (hoy por hoy es el de D'Hondt) se aplica a cada provincia por separado. Esa es la razón por la que algunos partidos con una presencia mayoritaria en zonas concretas consiguen proporcionalmente (al número de votos) más escaños en el Congreso que otros cuyos votos se reparten por toda España.

Existen otras alternativas a la Ley de D'Hondt que se diferencian en la forma de realizar los cocientes. Puede verse una comparación entre algunos de estos métodos (incluida la circunscripción única en la que no hay reparto de escaños por provincias) en el siguiente enlace:

<http://www.europapress.es/nacional/noticia-reforma-ley-electoral-asi-funcionan-otros-sistemas-20180208170220.html>

Finalidad: Recorrido de las componentes de un vector y usar vectores auxiliares. Dificultad Baja.

13. [Contiene débil] (Examen Enero 2018) Dados dos vectores grande y peque de tipo char, queremos comprobar si el primero contiene al segundo de la siguiente forma: todos los caracteres de peque tienen que aparecer en grande en el mismo orden, aunque no tienen por qué estar consecutivos. Por ejemplo, el vector grande = {'d','e','s','t','i','n','o'} contiene débilmente al vector peque = {'s','i'} pero no a peque = {'i','s'}.

Construya un programa que lea desde teclado los caracteres del vector grande, parando la entrada cuando se introduzca el carácter #. Haga lo mismo para introducir los caracteres del vector peque. El programa indicará si el vector grande contiene o no al vector peque.

Para leer los caracteres (incluido el espacio en blanco), haga uso de la instrucción `caracter = cin.get()` (en vez de `cin >> caracter`) tal y como se indica en la página Problemas-67.

Si lo desea, puede usar el esbozo del programa que puede encontrar en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/III_ContieneDebilEsbozo.cpp

Ejemplo de entrada: destino#si#

— Salida correcta: Sí lo contiene

Ejemplo de entrada: destino#is#

— Salida correcta: No lo contiene

Ejemplo de entrada: destino#no#

— Salida correcta: Sí lo contiene

Finalidad: Recorrido de las componentes de un vector. Dificultad Media.

14. **[Elimina un bloque (versión ineficiente)]** Sobre un vector de tipo char, se quiere eliminar todos los caracteres que haya entre dos posiciones. Por ejemplo, después de eliminar el bloque que hay entre las posiciones 1 y 3 del vector {'S', 'o', 'Y', ' ', 'y', 'o'}, ésta debe quedarse con {'S', 'y', 'o'}.

Un primer algoritmo para resolver este problema sería el siguiente (en ejercicios posteriores se verán métodos más eficientes):

Para borrar el bloque entre izda y dcha:

```
Recorrer cada componente -i- del vector
entre las posiciones izda y dcha
Eliminar dicha componente -i-
```

Para eliminar una posición hay que desplazar hacia la izquierda todas las componentes que hay a su derecha.

Construya un programa que lea caracteres hasta el terminador #. A continuación lea dos enteros que representen las posiciones izquierda (≥ 0) y derecha (entre izquierda y la última componente del vector) e imprima el vector resultante de quitar el bloque de caracteres.

Ejemplo de entrada: abcdefg# 1 3 — Salida correcta: aefg

Ejemplo de entrada: abcdefg# 1 0 -1 15 3 — Salida correcta: aefg

Ejemplo de entrada: abcdefg# 0 5 — Salida correcta: g

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

15. **[Elimina un bloque (versión eficiente)]** Resuelva el ejercicio 14 **[Elimina un bloque (versión ineficiente)]** pero de una forma eficiente. Para ello, implemente el siguiente algoritmo:

Para borrar el bloque entre izda y dcha:

```
Calcular num_a_borrar como dcha - izda + 1
```

```
Recorrer las componentes -i- del vector
```



```
entre las posiciones dcha+1 hasta el final
Colocar la componente -i- en la posición
i - num_a_borrar
```

Este algoritmo resuelve el problema con un único bucle mientras que la versión ineficiente recurría a dos bucles anidados.

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

16. [Elimina ocurrencias de una componente -versión ineficiente-] Se desea eliminar todas las apariciones de un determinado carácter `a_borrar`, dentro de un vector de caracteres. Por ejemplo, después de eliminar el carácter `'o'` del vector `{'S','o','Y',' ','y','o'}`, éste debe quedarse con `{'S','Y',' ','y'}`. Un primer algoritmo para resolver este problema sería el siguiente (en ejercicios posteriores se verán métodos más eficientes):

```
Recorrer todas las componentes del vector
Si la componente es igual al carácter a_borrar, eliminarla
(desplazando hacia la izda las componentes que hay a su dcha)
```

A la hora de implementar el anterior código, debe prestar especial atención cuando hay dos caracteres a borrar consecutivos.

Construya un programa que lea caracteres hasta que se introduzca `#` lo que formará el vector. A continuación lea el carácter a eliminar. El programa imprimirá el vector resultante después de eliminar todas las apariciones de dicho carácter.

Ejemplo de entrada: `maaaovaiala#a`

— Salida correcta: `movil`

Ejemplo de entrada: `aaaaa#a`

— Salida correcta: *vector vacío*

Finalidad: Recorrido de un vector eliminando componentes. Dificultad Baja.

17. [Elimina ocurrencias de una componente -versión eficiente-] Recupere la solución del ejercicio 16 [Elimina ocurrencias de una componente -versión ineficiente-]. Puede usar la solución que se encuentra en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/III_EliminaOcurrenciasIneficiente.cpp

Se pida que resuelva el mismo problema pero de una forma eficiente. Para que aprecie la ineficiencia de la solución anterior, descargue el fichero que se encuentra en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/Quijote_con_ruido.txt

Este fichero contiene el texto del Quijote pero con *ruido* en el sentido de que contiene numerosas ocurrencias del carácter `'~'` (código ASCII 126). Lo que queremos hacer

es eliminar todos esos caracteres aplicando la solución ineficiente anterior. Para poder ejecutar el programa debe tener en cuenta lo siguiente:

- El texto del Quijote viene preparado para que la lectura de datos sea tal y como se indica en el ejercicio 16 [Elimina ocurrencias de una componente -versión ineficiente-], es decir, en primer lugar aparecen todos los caracteres que forman el Quijote y al final habrá un carácter # como terminador de entrada de datos. Justo después aparece el carácter que queremos eliminar (en nuestro caso '~')
- Como va a almacenar todo el texto del Quijote en un vector de caracteres, debe dimensionar éste con una constante grande, por ejemplo 2e7. Al ser un vector muy grande, recuerde compilar el programa con las instrucciones detalladas en la página Problemas-68.

Si lo desea, puede usar el esbozo del programa que puede encontrar en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/III_EliminaOcurrenciasEficienteEsbozo.cpp

- Tendrá que redirigir la entrada de datos para que el programa lea los valores del fichero. Para ello, siga las indicaciones de la página 40. También debe redirigir la salida de datos para que el programa construya un fichero nuevo Quijote_sin_ruido.txt. En definitiva, desde línea de comando de la consola deberá poner:

```
NombreFichero.exe < Quijote_con_ruido.txt > Quijote.txt
```

Si ejecuta el programa del ejercicio 16 [Elimina ocurrencias de una componente -versión ineficiente-] observará que la ejecución puede tardar más de 12 minutos!

Para resolver eficientemente este problema se propone utilizar dos variables, pos_lectura y pos_escritura que nos vayan indicando, en cada momento, la componente que se está leyendo y el sitio dónde tiene que escribirse. Por ejemplo, supongamos que en un determinado momento la variable pos_lectura vale 6 y pos_escritura 3. Si la componente en la posición 6 es el carácter a borrar, simplemente avanzaremos pos_lectura. En caso contrario, colocaremos dicha componente en la posición 3 y avanzaremos una posición ambas variables.

Implemente este algoritmo y observe la diferencia de tiempo al ejecutarlo sobre el Quijote, ya que ahora el tiempo de ejecución es de unos 8 milisegundos.

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

18. [Sustituir carácter por vector (versión ineficiente)] Resuelva el ejercicio 4 [Sustituir carácter por vector (con vector auxiliar)] sin utilizar el vector auxiliar sustituido. Por lo tanto habrá que modificar el mismo vector de partida v con el resultado de sustituir todas las apariciones del carácter a_borrar por el otro vector de caracteres nuevo. Para ello, implemente el siguiente algoritmo:

Algoritmo para sustituir las apariciones de "a_borrar"
dentro de "v", por un vector "nuevo"

```
Recorrer las componentes i de v
  Si v[i] == a_borrar
    Elimina la componente i
    (desplazando a la izda las componentes que hay a su dcha)

    Inserta todas y cada una de las componentes de nuevo
    (desplazando a la dcha las componentes que hay
     a partir de la posición de inserción)
```

Se recomienda usar un bucle `while` para recorrer el vector principal `v`, ya que conforme se vayan introduciendo componentes, va cambiando el número de componentes utilizadas de `v` y, por lo tanto, no debe usar un bucle `for`.

Si lo desea, puede usar el esbozo del programa que puede encontrar en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/III_SustituyeCaracterVectorEsbozo.cpp

Finalidad: Recorrido de un vector eliminando e insertando componentes. Dificultad Media.

19. [Sustituir carácter por vector (versión eficiente)] (*Examen Febrero 2017*) Ejecute el programa pedido en el ejercicio 18 [Sustituir carácter por vector (versión ineficiente)] con el fichero del Quijote que se encuentra en:

https://decsai.ugr.es/jccubero/FP/Quijote_replace_blanco.txt

Debe redirigir la entrada de datos para que use dicho fichero y redirigir la salida de resultados para crear un fichero de texto nuevo, tal y como se vio en la página 40

El fichero anterior contiene los datos en el orden indicado en el ejercicio 18 [Sustituir carácter por vector (versión ineficiente)], es decir, primero todos los valores del vector (el texto en sí del Quijote), luego un carácter #, los caracteres del vector nuevo, otro carácter # y finalmente el carácter a borrar.

Al ser un vector muy grande, recuerde compilar el programa con las instrucciones detalladas en la página Problemas-68.

En este ejemplo, el vector `a_insertar` está formado por tres espacios en blanco y el carácter `a_borrar` es el espacio en blanco. Así pues, lo que pretendemos es crear un nuevo texto del Quijote con más espacio entre las palabras (3 espacios en vez de 1). Como queremos leer los espacios en blanco, la lectura de los caracteres debe hacerse con `cin.get()`.

Si ejecuta la solución planteada en el ejercicio 18 [Sustituir carácter por vector (versión ineficiente)] el programa tardará unos 45 minutos (sobre un i7 con 16Gb de RAM).

Por lo tanto, se pide implementar un algoritmo más eficiente que no necesite realizar tantos desplazamientos de las componentes del vector. Para ello, se recomienda seguir una aproximación distinta. La idea es ir colocando directamente cada componente en el sitio que le corresponde. Por ejemplo, si el vector v sólo contuviera una única ocurrencia de `a_borrar` (en la posición `pos_a_borrar`) y el vector nuevo contuviese 4 caracteres, bastaría colocar todas las componentes de v que hubiese después de `pos_a_borrar`, un total de 3 posiciones por encima. Una vez hecho este desplazamiento, bastaría colocar las 4 componentes de `nuevo`, a partir de `pos_a_borrar`.

Como puede haber más de una ocurrencia de `a_borrar` dentro de v , debe realizar un primer recorrido sobre todo el vector v para contar el número de ocurrencias de `a_borrar`.

Si resuelve correctamente este ejercicio, el tiempo de ejecución baja drásticamente a unos *0.2 segundos*.

En este ejercicio no puede usar ningún vector auxiliar.

Si lo desea, puede usar el esbozo del programa que puede encontrar en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/III_SustituyeCaracterVectorEsbozo.cpp

Finalidad: Trabajar eficientemente con vectores. Dificultad Media.

20. **[Top k (versión ineficiente)]** Se dispone de una serie de datos enteros positivos (de tipo `long`) y se quiere calcular los k mayores, ordenados de mayor a menor.

Construya un programa que vaya leyendo datos desde teclado hasta que se introduzca cualquier número negativo. A continuación lea el número k y calcule los k mayores.

Se le pide que aplique el siguiente algoritmo:

```
Vector original: v
Vector que contendrá los k mayores valores: topk

Copiar v en topk
Ordenar topk de MAYOR a MENOR <-- Atención!!!
(se recomienda modificar el algoritmo de ordenación
 por inserción)

Seleccionar los k primeros elementos de topk
```

Finalmente, imprima los k primeros valores del vector `topk` en pantalla.

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

— Salida correcta: 12 3

Finalidad: Ordenación de un vector y usar vectores auxiliares. Dificultad Baja.

21. [Top k (versión eficiente)] (*Examen Febrero 2015*) Ejecute la solución del ejercicio 20 [Top k (versión ineficiente)] con el siguiente fichero (con cien mil valores) como entrada de datos (debe redirigir la entrada de datos para que use dicho fichero tal y como se vio en la página 40):

https://decsai.ugr.es/jccubero/FP/datos_topk1e5.txt

Tenga en cuenta las modificaciones que tiene que hacer en el entorno de compilación para poder trabajar con vectores grandes (ver página Problemas-68 al inicio de esta Relación de Problemas)

La salida debe ser:

999954038 999948623 999919447 999912553 999909127

En un ordenador i7 tardará unos 4 segundos en ejecutarse. Si lo hacemos con el siguiente fichero (con un millón de datos):

https://decsai.ugr.es/jccubero/FP/datos_topk1e6.txt

cuya salida debe ser:

999999875 999999069 999997647 999997024 999995114

tardará unos *9 minutos!*.

Por lo tanto, se pide resolver el ejercicio 20 [Top k (versión ineficiente)] usando un algoritmo más eficiente. Para ello, observe que no es necesario ordenar todo el vector. Basta con ordenar al principio k valores y posteriormente, ir insertando de forma ordenada el resto de valores del vector.

El nuevo algoritmo ha de ser capaz de ordenar los cien mil datos en medio segundo y tardar no más de cuatro segundos en ordenar el fichero con un millón de datos.

Finalidad: Ordenación de un vector. Dificultad Media.

Problemas sobre matrices

22. [Traspuesta] Lea desde teclado dos variables `util_filas` y `util_columnas` y lea los datos de una matriz de reales de tamaño `util_filas` x `util_columnas`. Sobre dicha matriz, se pide lo siguiente:

Calcular la traspuesta de la matriz, almacenando el resultado en otra matriz y mostrar el resultado.

Ejemplo de entrada:

```
3 4
9    7    4    5
2    18   2   12
7    9    1    5
```

— — Salida correcta:

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

Finalidad: Recorrido de las componentes de una matriz. Dificultad Baja.

23. [Producto de matrices] Lea los datos de una matriz de reales tal y como se indica en el ejercicio 22 [Traspuesta] de esta relación de problemas.

Si la matriz que se ha introducido es $n \times m$, por ejemplo, ahora se procederá a leer los datos de una segunda matriz $m \times k$. Por lo tanto, en primer lugar se lee el entero k y a continuación los valores de esta segunda matriz.

Multiplique ambas matrices y muestre el resultado en pantalla.

Si lo desea, puede usar el esbozo del programa que puede encontrar en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/III_ProductoMatricesEsbozo.cpp

Ejemplo de entrada:

```
3 4
9    7    4    5
2    18   2   12
7    9    1    5
2
```

1 2
3 4
5 6
7 8

— — Salida correcta:

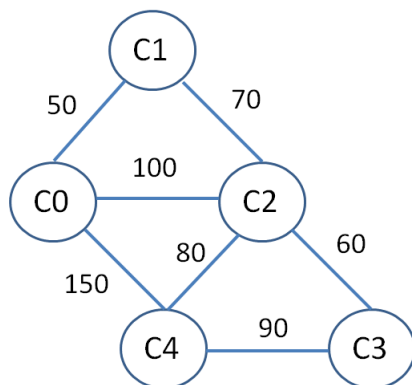
85 110
150 184
74 96

Finalidad: Recorrido de las componentes de una matriz. Dificultad Baja.

24. **[Distancias entre ciudades]** Se desea construir una matriz de `double` para almacenar y gestionar las distancias de los caminos directos que conectan un conjunto de ciudades.

Si entre dos ciudades no existe un camino directo, se almacenará un cero. Se supone que la distancia de una ciudad consigo misma será cero y que las distancias son simétricas.

Un ejemplo con 5 ciudades sería:



	C0	C1	C2	C3	C4
C0	0	50	100	0	150
C1	50	0	70	0	0
C2	100	70	0	60	80
C3	0	0	60	0	90
C4	150	0	80	90	0

Sabemos que esta representación de las distancias no es óptima ya que la matriz es simétrica y por tanto la mitad de los datos están duplicados. Pero asumimos esta aproximación para simplificar el problema. Suponga que nunca se trabajará con más de 50 ciudades.

Use el esbozo del programa que puede encontrar en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/III_CiudadesEsbozo.cpp

En este fichero se encuentra el esbozo del programa principal que realiza la lectura de los datos de la matriz. Puede observar que el programa principal lee los datos en el siguiente orden:

RELACIÓN DE PROBLEMAS DEL TEMA III. Vectores y Matrices

- a) En primer lugar lee el número de filas. Este valor, al ser una matriz cuadrada, coincide con el número de columnas.
- b) A continuación lee los datos de la sub-matriz diagonal superior

Según el ejemplo anterior, los datos a introducir serían:

```
5
50  100  0   150
    70   0   0
        60  80
            90
```

Observe que, en el ejemplo anterior, el número de filas es 5. A continuación se leen los datos de la matriz diagonal superior. Para la primera fila, se leen 4 datos (50 100 0 150) y no 5 ya que el primero es cero, al estar en la diagonal principal (la distancia de una ciudad consigo misma es cero) Los datos siguientes (70 0 0) corresponden a la segunda fila, y así sucesivamente.

Las ciudades vendrán determinadas por un índice, a partir de cero.

Se pide obtener la ciudad con mayor número de conexiones directas. En el ejemplo anterior, la ciudad con más conexiones es la ciudad 2 con 4 conexiones.

Observe que, en este ejercicio, no hemos usado los valores de las distancias (sólo si son distintas de cero) En el ejercicio 25 [Distancias entre ciudades (mejor escala)] sí se usarán.

Ejemplo de entrada:

```
5
50  100  0   150
    70   0   0
        60  80
            90
```

— — Salida correcta:

La ciudad más conectada es la ciudad 2 con 4 conexiones

Finalidad: Trabajar con una matriz. Dificultad Media.

25. [Distancias entre ciudades (mejor escala)] Recupere la solución del ejercicio 24 [Distancias entre ciudades]. Se pide que, después de leer los datos de las distancias de las ciudades, el programa lea los índices i y j de dos ciudades. En el caso de que no exista un camino directo entre ellas (la distancia entre ellas sea cero), el programa debe obtener aquella ciudad intermedia o *escala* z que permita hacer el trayecto entre i y j de la forma más económica posible. Es decir, se trata de encontrar una ciudad z tal que $d(i, z) + d(z, j)$ sea mínima ($d(a, b)$ es la distancia entre las ciudades a y b).

Por ejemplo, si se desea viajar desde la ciudad 1 a la 4, hacerlo a través de la ciudad 0 tiene un costo de $d(1, 0) + d(0, 4) = 50 + 150 = 200$ mientras que si se hace a través de la ciudad 2, el costo sería $d(1, 2) + d(2, 4) = 70 + 80 = 150$.

Nota: Para la inicialización del mínimo, puede usar la constante `INFINITY`, que garantiza que cualquier dato de tipo `double` es menor que ella.

Use el esbozo del programa que puede encontrar en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/III_CiudadesEsbozo2.cpp

Ejemplo de entrada:

```
5
50  100  0   150
    70   0   0
        60  80
            90
1  4
```

— Salida correcta:

```
La mejor escala entre las ciudades 1 y 4 es la ciudad 2
```

Otro ejemplo de mejor escala:

```
La mejor escala entre las ciudades 0 y 3 es la ciudad 2
```

Finalidad: Trabajar con una matriz. Dificultad Media.

26. [Asignación de pedidos a técnicos] (*Examen Enero 2018*) Una empresa de servicios dispone de n técnicos y n pedidos por atender. La empresa dispone de una matriz $T^{n \times n}$ de tarifas donde cada t_{ij} indica el precio que cobra el técnico i por atender el pedido j . Supondremos que t_{ij} es un entero mayor estricto que 0 y menor o igual que 100.

Se desea asignar cada técnico a un pedido. Una asignación se puede modelizar mediante una matriz de `bool` $A^{n \times n}$ donde $A_{ij} = \text{true}$ significa que el técnico i atiende el pedido j , y $A_{ij} = \text{false}$ en caso contrario (ver figura de abajo). Una asignación válida es aquella en la que a cada técnico solo le corresponde un pedido y cada pedido está asignado a un único técnico. Por lo tanto, si una fila de A contiene `true` en una columna, ninguna otra fila debe contener `true` en dicha columna.

En la figura siguiente, se encuentra un ejemplo de T y A :

$$T = \begin{bmatrix} 21 & 12 & 31 \\ 16 & 14 & 25 \\ 12 & 18 & 20 \end{bmatrix} \rightarrow A = \begin{bmatrix} \text{false} & \text{true} & \text{false} \\ \text{true} & \text{false} & \text{false} \\ \text{false} & \text{false} & \text{true} \end{bmatrix}$$

RELACIÓN DE PROBLEMAS DEL TEMA III. Vectores y Matrices

Diremos que el coste total de una asignación válida dada por una matriz A es la suma de todas las asignaciones de cada técnico al pedido que se le haya asignado. En el ejemplo anterior, el coste total de la asignación es $12 + 16 + 20 = 48$.

Dada una matriz T el problema consiste en construir una matriz A que tenga un coste total lo más bajo posible.

Para ello, se pueden seguir varias estrategias. La más sencilla sería la siguiente:

- Para cada técnico, de entre los pedidos que no han sido asignados previamente a ningún otro técnico, le asignamos el más económico.
- Se empieza con cualquier técnico, por ejemplo, el primero.

Dada la matriz T de tarifas indicada arriba, empezaríamos con el técnico 0 (fila 0) y se le asigna el pedido de menor coste que es el 1 (columna 1, con coste 12). El técnico 1 se asigna al pedido 0, ya que, aunque el de menor coste es el del pedido 1 (con coste 14) éste ya ha sido asignado a un técnico anterior (en este caso, al técnico 0). Finalmente, el último técnico se asigna al único pedido que queda libre que es el 2.

Se pide construir un programa que lea el número n de técnicos (que es igual al de pedidos) y la matriz de tarifas T ($n \times n$ elementos). El programa calculará la matriz A de asignaciones usando la estrategia anterior e imprimirá a qué pedido es asignado cada técnico. El programa también calculará e imprimirá el coste total.

Nota. Observe que la estrategia indicada anteriormente no garantiza la obtención de una asignación óptima, es decir, con un coste total mínimo.

Si lo desea, puede usar el esbozo del programa que puede encontrar en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/III_AsignacionTecnicosPedidosEsbozo.cpp

Ejemplo de entrada:

```
3
21 12 31
16 14 25
12 18 20
```

— Salida correcta:

```
Técnico 0 -> Pedido 1
Técnico 1 -> Pedido 0
Técnico 2 -> Pedido 2
Coste total: 48
```

Finalidad: Recorrido de una matriz.

Dificultad Media.

RELACIÓN DE PROBLEMAS DEL TEMA III. Vectores y Matrices

27. [Matriz promedio] Defina dos matrices de reales *original* y *suavizada* de tamaño 50×30 . Lea desde teclado los valores de la matriz *original*, obligando a que sea cuadrada. Para ello, lea el número de filas n y a continuación introduzca los $n \times n$ datos de la matriz.

Construya la matriz *promedio* acorde a las siguientes indicaciones:

- La tabla resultante será **simétrica**.
- Los valores de la **diagonal principal** de la tabla resultante serán iguales a los de la tabla original.
- Los valores del **triángulo superior** de la tabla resultante se calculan de la siguiente manera: si (i, j) es una posición en el triángulo superior de la tabla resultante, su valor es la media aritmética de los valores que ocupan las posiciones de las columnas $j, j + 1, \dots, n - 1$ en la fila i de la tabla original.

Ejemplo de entrada:

```
4
9   7   4   5
2  18   2  12
7   9   1   5
0   1   2   3
```

— Salida correcta:

```
9      5.33333 4.5      5
5.33333 18      7      12
4.5      7      1      5
5      12      5      3
```

Finalidad: Recorrido de las componentes de una matriz. Dificultad Baja.

28. [Máximo de los mínimos] Lea los datos de una matriz de reales tal y como se indica en el ejercicio 22 [Traspuesta] de esta relación de problemas.

Si lo desea, puede usar el esbozo del programa que puede encontrar en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/III_MaximoMinimosEsbozo.cpp

Calcule la posición de aquel elemento que sea el mayor de entre los mínimos de cada fila. Por ejemplo, dada la matriz M (3×4),

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

el máximo entre 4, 2 y 1 (los mínimos de cada fila) es 4 y se encuentra en la posición (0, 2). Para ello, construya un vector con los mínimos de cada fila y luego recorra dicho vector para calcular el máximo.

Ejemplo de entrada: 3 4 9 7 4 5 2 18 2 12 7 9 1 5

— Salida correcta: Posición 0, 2. Valor: 4:

Finalidad: Recorrido de las componentes de una matriz. Dificultad Baja.

29. [Máximo de los mínimos (eficiente)] (*Examen Septiembre 2011*) Resuelva el mismo problema planteado en el ejercicio 28 [Máximo de los mínimos] pero de una forma más eficiente. Para ello, no va a necesitar almacenar en un vector todos los mínimos de cada fila.

Supongamos que en una variable `max` tenemos almacenado el máximo (de los mínimos de las filas) de las n primeras filas. Si empezamos a recorrer la fila $n + 1$ y nos encontramos con una casilla que es menor que `max`, entonces podemos parar el recorrido de dicha fila y no llegar hasta el final de la misma ya que el mínimo final de dicha fila será menor que `max` y por tanto no lo actualizará.

Finalidad: Recorrido de las componentes de una matriz. Dificultad Media.

Ejercicios complementarios

30. **[Elimina varios]** Se quiere eliminar un conjunto de posiciones `a_borrar` de un vector `v` de enteros. Por ejemplo, si el vector contiene `-3 1 15 6 4 -1 8`, después de eliminar el conjunto de posiciones dado por `2 5 3`, el vector se quedará con `-3 1 4 8`. El conjunto de posiciones puede venir dado en cualquier orden.

Observe que una posibilidad sería sustituir los enteros a borrar por un entero especial, por ejemplo 0 y luego pasarle un algoritmo que eliminase todas las ocurrencias de 0. Sin embargo, debemos evitar esta técnica ya que no podemos presuponer que tenemos la posibilidad de elegir un entero especial en todas las situaciones posibles.

Se recomienda implementar el siguiente algoritmo:

Ordenar `a_borrar` (el vector de posiciones)

Utilizar dos índices: `pos_escritura` y `pos_lectura` que marquen las posiciones de lectura y escritura en el vector `v`

Recorrer con `pos_lectura` los caracteres del vector `v`

Si el carácter actual no está en una posición a borrar, colocarlo en `pos_escritura`.

Construya un programa principal que lea el número de componentes a rellenar del vector `v` y a continuación los valores de éste. A continuación lea el tamaño del vector `a_borrar` y sus componentes. El programa borrará las componentes indicadas y mostrará el resultado en pantalla.

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

— Salida correcta: -3 1 4 8

Finalidad: Ordenación de un vector y eliminar eficientemente. Dificultad Media.

31. **[Metapentagonales]** (*Examen Febrero 2020*) Un entero se dice que es pentagonal si puede obtenerse a partir de la fórmula $\frac{n(3n-1)}{2}$ para algún valor de $n > 0$. Los primeros 10 números pentagonales son 1, 5, 12, 22, 35, 51, 70, 92, 117 y 145. Se dice que una pareja de números pentagonales p_i, p_j es metapentagonal si la media (entera) entre ambos $\frac{p_i + p_j}{2}$ y la diferencia de ambos en valor absoluto $abs(p_i - p_j)$ también son números pentagonales. Observe que ambos valores son **enteros** mayores que cero y menores que el máximo entre p_i y p_j .

Se pide encontrar la pareja de números metapentagonales más alejada entre sí, es decir, aquella pareja (p_i, p_j) cuya diferencia en valor absoluto $abs(p_i - p_j)$ sea máxima.

RELACIÓN DE PROBLEMAS DEL TEMA III. Vectores y Matrices

El programa leerá un entero *tope* que será el número de enteros pentagonales a considerar (imponga en la lectura que $0 < \text{tope} \leq 1000$) e imprimirá la pareja de números metapentagonales más lejana.

Para resolver este problema podría simplemente despejar en la fórmula de los números pentagonales y calcular la pareja más lejana. Sin embargo, aunque sea más ineficiente, se pide que implemente el siguiente algoritmo:

```
Almacene en un vector p los primeros tope pentagonales
  Recorra todas las parejas (izda y dcha) de p entre 0 y tope
    Compruebe si la media y la diferencia entre izda y dcha
      están en p (son pentagonales)
    En caso afirmativo, compruebe si la distancia entre ellos
      es mayor que la máxima hasta el momento
```

Ejemplo de entrada: 1000 — Salida correcta: 28912 838882

Finalidad: Bucles anidados para buscar dentro de un vector. Dificultad Media.

32. **[Contiene de forma cíclica]** (*Examen Febrero 2013. Doble grado*) Se quiere ver si un vector de caracteres *grande* contiene todos los caracteres de otro vector *peque* en el mismo orden (no tienen que estar consecutivos) y de forma *cíclica*. Para que se cumpla este criterio se debe satisfacer que todos los caracteres de *peque* estén en *grande*, en el mismo orden aunque no de forma consecutiva. Además, si durante la búsqueda se ha llegado al final del vector *grande*, se debe proseguir la búsqueda por el inicio de *grande*, pero sin sobrepasar la posición en la que hubo la primera concordancia. Por ejemplo, los siguientes vectores contienen a *peque* = {abcd}

```
x a y o b c p d g
c p d g x a y o b
y o b c p d g x a
```

Hay que destacar que la primera letra de *peque* a buscar en *grande* podría estar en cualquier sitio. Por ejemplo, si *grande* = bcada y *peque* = abcd, podemos ver que a partir de la primera a de *grande* no podemos encontrar *peque* de forma cíclica ya que al ciclar, no podemos seguir buscando más allá de la aparición de la primera a. Sin embargo, si empezamos a buscar a partir de la segunda aparición del carácter a de *grande*, podemos ver que sí encontramos todos los caracteres.

Finalidad: Recorrido de las componentes de un vector. Dificultad Media.

33. **[Series ascendentes]** (*Examen Septiembre Doble Grado 2013*) Se quiere calcular el número de series ascendentes de un vector de caracteres. Por ejemplo, el vector ttuvghtwwbde tiene 3 series ascendentes que son ttuv, ghtww, bde.

Construya un programa que lea un conjunto de caracteres con terminador #, calcule e imprima el número de series ascendentes.

Ejemplo de entrada: ttuvghtwwbde# — Salida correcta: 3

Ejemplo de entrada: gfed# — Salida correcta: 4

Ejemplo de entrada: # — Salida correcta: 0

Finalidad: Recorrido sobre un vector procesando dos componentes en cada iteración. Dificultad Media.

34. **[Login]** Se está diseñando un sistema web que recolecta datos personales de un usuario y, en un momento dado, debe sugerirle un nombre de usuario (login). Dicho login estará basado en el nombre y los apellidos; en concreto estará formado por los N primeros caracteres de cada nombre y apellido (en minúsculas, unidos y sin espacios en blanco). Por ejemplo, si el nombre es "Antonio Francisco Molina Ortega" y $N = 2$, el nombre de usuario sugerido será "anfrmoor".

Debe tener en cuenta que el número de palabras que forman el nombre y los apellidos puede ser cualquiera. Además, si N es mayor que alguna de las palabras que aparecen en el nombre, se incluirá la palabra completa.

Por ejemplo, si el nombre es "Ana CAMPOS de la Blanca" y $N=4$, entonces la sugerencia será "anacampdelablan" (observe que se pueden utilizar varios espacios en blanco para separar palabras).

Construya un programa que lea caracteres con `cin.get()` hasta llegar al terminador `'>'` y los vaya introduciendo en una variable `a_codificar` de tipo de dato `string`. Para ello debe usar `a_codificar.push_back(caracter)` tal y como se indica en las transparencias de clase. A continuación, el programa leerá el valor de N .

El programa debe construir otra variable codificada de tipo de dato `string` con el login correspondiente e imprimirlo en pantalla (también debe usar `codificada.push_back(caracter)`).

Ejemplo de entrada: Codillo Ificante Carlos>3

— Salida correcta: codificar

Finalidad: Recorrido de las componentes de un string, controlando qué ha ocurrido anteriormente. Dificultad Media.

35. **[k mayores que otro valor]** (*Examen Septiembre 2016*) Se dispone de un conjunto de reales positivos y se quiere construir otro vector `mayores_que` que contenga los k primeros valores que son mayores o iguales que otro valor de referencia. Los valores que se obtengan como resultado deben estar ordenados de menor a mayor.

Por ejemplo, si el vector original es

{5.2 2.5 7.3 4.2 3.1 4.9}

y el valor de referencia es 4.3, el vector `mayores_que` debe quedar así:

{4.9 5.2 7.3}

RELACIÓN DE PROBLEMAS DEL TEMA III. Vectores y Matrices

Construya un programa que vaya leyendo datos desde teclado e introdúzcalos en un vector. La entrada de datos termina con el -1. A continuación lea el real de referencia y el entero k . Calcule el vector `mayores_que` e imprímalo en pantalla.

Aplique el siguiente algoritmo:

Copiar el vector original en "mayores_que"

Ordenar el vector "mayores_que" de menor a mayor
(utilizar el algoritmo de ordenación por inserción)

Seleccionar los k primeros que sean mayores que la referencia

Utilice también el fichero de datos

https://decsai.ugr.es/jccubero/FP/mayores_que.txt

La salida correcta se encuentra en el fichero

https://decsai.ugr.es/jccubero/FP/mayores_que_solucion.txt

Finalidad: Ordenación de un vector. Dificultad Baja.

36. [**k mayores que otro valor, eficiente**] Modifique la solución del ejercicio 35 usando un algoritmo más eficiente. Observe que no hace falta ordenar todo el vector, sino únicamente considerar los datos que son mayores que la referencia.

Aplique el siguiente algoritmo:

Recorrer las componentes del vector original
Si es mayor que la referencia, insertar dicho
valor de forma ordenada en el vector "mayores_que"

El vector "mayores_que" siempre tendrá,
como mucho, k componentes

Mientras que la versión vista en el ejercicio 35 tardaba varios segundos, esta nueva versión tarda menos de un segundo.

Finalidad: Recorrido sobre un vector, insertando componentes. Dificultad Media.

37. [**Descodifica**] (*Examen Febrero 2016*) Dado un vector de caracteres que contiene un mensaje cifrado, se pide construir otro vector nuevo con el mensaje descifrado. La forma de descifrado consiste en coger la primera y última letra de cada palabra. Las palabras están separadas por uno o más espacios en blanco o el final del vector. Si la última palabra no tiene espacios en blanco a su derecha, se coge sólo el primer carácter.

Por ejemplo, si denotamos el inicio y final de la secuencia con un corchete, entonces:

[Hidrógeno limpia] se descodificaría como [Hola]

RELACIÓN DE PROBLEMAS DEL TEMA III. Vectores y Matrices

Reserve memoria para trabajar con un máximo de 1000 caracteres.

Para leer el mensaje cifrado debe leer caracteres con `cin.get()` (repase lo visto al inicio de esta relación de problemas) hasta que el usuario introduzca el carácter `#`. A continuación, el programa mostrará la cadena descodificada.

Ejemplo de entrada: [Hidrógeno limpia] -- Salida correcta: [Hola]

Ejemplo de entrada: [Hidrógeno limpia] -- Salida correcta: [Hol]

Ejemplo de entrada: [Hidrógeno] -- Salida correcta: [H]

Ejemplo de entrada: [Hidrógeno] -- Salida correcta: [Ho]

Ejemplo de entrada: [H] -- Salida correcta: [H]

Ejemplo de entrada: [H] -- Salida correcta: [H]

Finalidad: Recorrido de las componentes de un vector, controlando qué ha ocurrido anteriormente. Dificultad Media.

38. **[Histograma de vocales]** (*Examen Septiembre 2016*) Queremos calcular la frecuencia absoluta de las vocales (sin acentuar) presentes en un texto y mostrar el resultado en forma de un diagrama de barras en el que cada barra corresponde a una vocal y la altura representa la frecuencia de aparición de la correspondiente vocal.

Un ejemplo de diagrama de barras sería el siguiente:

```
6          *
5      *      *
4      *  *  *
3  *  *  *  *  *
2  *  *  *  *  *
1  *  *  *  *  *
    a  e  i  o  u
```

Cada aparición de una vocal la representaremos con un asterisco `*`. Por ejemplo, la barra correspondiente a la vocal `i` tiene un total de tres asteriscos ya que dicha vocal ha aparecido tres veces en el texto.

Defina una matriz de caracteres para representar las barras. Se emplea el carácter `'*'` para indicar que la casilla está ocupada y el carácter `' '` (espacio en blanco) para indicar que la casilla está libre. En la matriz no almacenaremos ni los números 1, 2, 3, 4, ... ni las vocales a, e, i, o, u.

Escriba un programa que lea una serie indefinida de caracteres de la entrada estándar (terminados en `#`) y muestre la figura (las barras, los conteos y las vocales) que representa la frecuencia absoluta de las vocales introducidas.

Por ejemplo, la anterior figura sería el diagrama de barras de la siguiente entrada:

```
You should solve this typical exam problem
in no more than 30 minutes#
```

RELACIÓN DE PROBLEMAS DEL TEMA III. Vectores y Matrices

Finalidad: Construir índices y trabajar con las componentes de una matriz. Dificultad Media.

39. [Maximin] Lea los datos de una matriz de reales tal y como se indica en el ejercicio 22 [Traspuesta] de esta relación de problemas.

Vea si existe un valor *MaxiMin*, es decir, que sea a la vez, máximo de su fila y mínimo de su columna.

Finalidad: Recorrido de las componentes de una matriz. Dificultad Media.

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

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

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

Finalidad: Recorrido de las componentes de una matriz. Dificultad Media.

41. Escribir un programa que permita a dos jugadores jugar al tres-en-raya. El programa preguntará por los movimientos alternativamente al jugador X y al jugador O. El programa mostrará las posiciones del juego como sigue:

1	2	3
4	5	6
7	8	9

Los jugadores introducen sus movimientos insertando los números de posición que desean marcar. Después de cada movimiento, el programa mostrará el tablero cambiado. Un tablero de ejemplo se muestra a continuación.

X	X	O
4	5	6
O	8	9

El programa detectará al final de la partida si hay o no empate y en caso contrario, qué jugador ha ganado. Además, pedirá empezar una nueva partida y reiniciar el proceso.

Finalidad: Practicar con el uso de matrices sencillas en una aplicación. Dificultad Media.

RELACIÓN DE PROBLEMAS DEL TEMA III. Vectores y Matrices

42. Escribir un programa para asignar asientos de pasajeros en un avión. Asumimos un avión pequeño con la numeración de asientos como sigue:

1	A	B	C	D
2	A	B	C	D
3	A	B	C	D
4	A	B	C	D
5	A	B	C	D
6	A	B	C	D
7	A	B	C	D

El programa mostrará con una X el asiento que está ya asignado. Por ejemplo, después de asignar los asientos 1A, 2B, y 4C, lo que se mostrará en pantalla tendrá un aspecto como este:

1	X	B	C	D
2	A	X	C	D
3	A	B	C	D
4	A	B	X	D
5	A	B	C	D
6	A	B	C	D
7	A	B	C	D

Después de mostrar los asientos disponibles, el programa pregunta por el asiento deseado, el usuario teclea un asiento y el programa actualiza la asignación mostrando el esquema anterior. Primero se pide el número de fila y después la letra de asiento. Esto continua hasta que todos los asientos se asignen o hasta que el usuario indique que no quiere asignar más asientos (introduciendo el valor -1 en el número de la fila). Si el usuario introduce un asiento ya asignado, el programa mostrará un mensaje indicando que el asiento está ocupado y volverá a solicitarlo.

Finalidad: Practicar con el uso de matrices sencillas en una aplicación. Dificultad Baja.

43. [Límites de velocidad -en una matriz-] Recupere la solución del ejercicio 102 [Cuadro de límites de velocidad] de la Relación de Problemas II. Modifique el programa para separar E/S y C. Para ello, defina una matriz y guarde los valores de los límites en dicha matriz. Una vez se tenga la matriz de límites, imprímala en pantalla.

Finalidad: Recorrido de las componentes de una matriz. Dificultad Baja.

44. [Viajante de comercio] (Examen Febrero Extraordinario 2018) Un diligente agente de comercio tiene que visitar periódicamente a sus n clientes. Cada uno de ellos vive en una ciudad distinta. Su jefe le ordena que prepare el plan de viaje del mes siguiente según las siguientes reglas:

- Tiene que visitar a todos los clientes y una sola vez a cada uno.

RELACIÓN DE PROBLEMAS DEL TEMA III. Vectores y Matrices

- El recorrido se hará de forma que el coste total sea el menor posible.
- No importa por qué ciudad comience, pero debe finalizar en la misma ciudad en la que comenzó. Por tanto, el recorrido forma un ciclo. Y, obviamente, no puede volver a la ciudad inicial hasta haber visitado todas las demás.

Como datos de entrada se facilitan el número de clientes, n , y el coste de viaje que hay entre cada par de ciudades, expresada como una matriz, C , de tamaño $n \times n$, donde C_{ij} es el coste de viajar desde la ciudad i a la j (dicha matriz no tiene por qué ser simétrica). Supondremos que todos estos valores son positivos, $C_{ij} > 0$ y que se introducen correctamente.

Este es un problema muy difícil de resolver para el que existen distintas soluciones aproximadas. Uno de esos algoritmos, llamado *heurística del vecino más cercano*, funciona así: se selecciona una ciudad de partida, a , y se busca la ciudad más económica para llegar, b . A continuación, se busca la ciudad *aún no visitada* más económica a b . Y así hasta completar la visita a todas las ciudades. El coste del recorrido es la suma de todos los costes intermedios.

Implemente el algoritmo descrito anteriormente. Construya un programa que lea el valor de n y los valores de la matriz C , así como la ciudad inicial. Luego calculará el recorrido y posteriormente su coste. Finalmente mostrará ambos resultados por la salida estándar.

Por ejemplo, supongamos un problema con $n = 4$ cuya matriz de costes es:

$$C = \begin{bmatrix} - & 10 & 15 & 9 \\ 11 & - & 13 & 8 \\ 17 & 21 & - & 15 \\ 26 & 7 & 14 & - \end{bmatrix}$$

Por simplicidad, asumimos que los nombres de las ciudades son 0, 1, ..., n . Si partimos de la ciudad 0, entonces la solución encontrada es: 0, 3, 1, 2 con un coste de $9 + 7 + 13 + 17 = 46$. Si el viaje comienza por la ciudad 2, el recorrido final sería 2, 3, 1, 0 con un coste de $15 + 7 + 11 + 15 = 48$.

Ejemplo de entrada:

4 10 15 9 11 13 8 17 21 15 26 7 14 2

— Salida correcta:

Recorrido: 2 3 1 0 - Coste: 48

Finalidad: Recorrer los datos de una matriz usando un vector auxiliar de elementos ya visitados. Dificultad Media.

RELACIÓN DE PROBLEMAS DEL TEMA IV. Funciones y Clases

Problemas sobre funciones

1. [Errores en funciones] Encuentre los errores, si los hubiese, de las siguientes funciones. Indique qué haría para que funcionasen correctamente. No es necesario que construya ningún programa. Entregue únicamente un fichero de texto.

```
int Sumale1(){
    return entero + 1;
}
int Sumale1(int entero){
    entero = entero + 1;
}
int ValorAbsoluto(int entero){
    if (entero < 0)
        entero = -entero;
    else
        return entero;
}
bool EsPositivo(int valor){
    if (valor > 0)
        return true;
}
long ParteEntera(double real){
    int parte_entera;

    parte_entera = trunc(real);

    return parte_entera;
}
```

Considere ahora la siguiente función y la llamada en el programa principal:

```
int Sumale1(int entero){
    return entero + 1;
}
int main(){
    int n = 5;
    Sumale1(n);
}
```

```
    cout << n;  
}
```

El programa compila, pero imprime en pantalla 5 en vez de 6. ¿Por qué? ¿Qué solución propone?

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

2. **[Potencia entera]** Defina una función `PotenciaEntera` para elevar un número real a otro entero. En la implementación, no puede usar la función `pow` de `cmath`, sino que debe hacerlo con un bucle, tal y como se vio en el ejercicio 23 **[Factorial y Potencia]** de la Relación de Problemas II. Escriba un programa simple de prueba.

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

3. **[Valores de la Gaussiana]** Recupere la solución del ejercicio 21 **[Valores de la Gaussiana]** de la relación de problemas II. Puede usar la solución propuesta en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/II_Gaussiana.cpp

Reescriba la solución al problema definiendo la función `Gauss` para que calcule el valor de la gaussiana en un valor de abscisa x (dado un valor de esperanza y desviación típica)

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

4. **[Mismo signo (multiplicando)]** Recupere la solución del ejercicio 1 **[Mismo signo - multiplicando-]** de la Relación de Problemas II cuyo código puede encontrar en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/II_MismoSignoMultiplicando.cpp

Modifique dicho código para que el programa principal llame a una función `MismoSigno` que nos dice si dos enteros tienen o no el mismo signo.

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

5. **[Interés bancario (capital reinvertido)]** Recupere la solución del ejercicio 24 **[Interés bancario (capital reinvertido)]** de la Relación de Problemas II cuyo código puede encontrar en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/II_InteresReinvierte.cpp

Modifique dicho código para que el programa principal llame a una función `CapitalAcumulado` que calcule el capital obtenido tras aplicar un número de años

un interés dado. El programa NO imprimirá el capital acumulado cada año. Sólo imprimirá el capital obtenido al final.

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

6. [Interés bancario (doblar)] Recupere la solución del ejercicio 25 [Interés bancario (doblar)] de la Relación de Problemas II cuyo código puede encontrar en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/II_InteresDoblar.cpp

Modifique dicho código para que el programa principal llame a una función NumAñosHastaObtener. El programa no debe imprimir el capital obtenido cada año. Solamente debe imprimir el número de años que han de transcurrir hasta obtener el doble del capital inicial.

¿Es mejor una función para calcular el número de años hasta obtener el doble de la cantidad inicial, o por el contrario es mejor una función para calcular el número de años hasta obtener una cantidad específica?

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

7. [Grados a radianes] Recupere la solución del ejercicio 9 [Conversión de grados a radianes] de la Relación de Problemas I cuyo código puede encontrar en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/I_GradosToRadianes.cpp

Modifique dicho código para que el programa principal llame a una función ToRadianes que haga la conversión de grados a radianes.

Finalidad: Familiarizarnos con la definición de funciones. Uso de constantes globales. Dificultad Baja.

8. [Diferencia entre instantes con funciones] Recupere la solución del ejercicio 13 [Segundos entre dos instantes] de la Relación de Problemas I cuyo código puede encontrar en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/I_DiferenciaInstantes.cpp

Modifique dicho código para que el programa principal llame a una función SegundosEntre que calcule la diferencia en segundos entre dos instantes (cada instante viene determinado por la hora, minuto y segundo)

Finalidad: Familiarizarnos con la definición de funciones. Uso de constantes globales. Dificultad Baja.

9. [Comparación entre instantes con funciones] Recupere la solución del ejercicio 5 [Comparación de dos instantes (restando segundos)] de la Relación de Problemas II cuyo código puede encontrar en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/II_CompararInstantesRestando.cpp

Modifique dicho código para que el programa principal llame a una función `EsMenor`. Dicha función nos indicará si un instante es menor que otro. Para implementar dicha función, debe llamar a la función `SegundosEntre` definida en el ejercicio 8 [Diferencia entre instantes con funciones] de esta Relación de Problemas.

Finalidad: Familiarizarnos con la definición de funciones y las llamadas entre ellas. Dificultad Baja.

10. [Máximo de 3 valores] Defina la función `Max2` para que calcule el máximo de dos valores enteros. Defina ahora la función `Max3` para que calcule el máximo entre tres valores enteros. En la implementación de la función `Max3`, ésta debe llamar a la función `Max2`.

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

Ejemplo de entrada: -2 7 5

— Salida correcta: El máximo es: 7

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

11. [Decimal redondeado] Recupere la solución del ejercicio 10 [Decimal redondeado] de la Relación de Problemas I cuyo código puede encontrar en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/I_Round.cpp

Modifique dicho programa para que el programa principal llame a una función `Redondea`. Esta función debe redondear el real a un número de cifras decimales. En la implementación de esta función, necesita calcular una potencia entera. Para ello, llame a la función `PotenciaEntera` definida en el ejercicio 2 [Potencia entera] y cuyo código pueden encontrar en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/IV_FactorialPotencia.cpp

Ejemplo de entrada: 3.49 1 — Salida correcta: 3.5

Ejemplo de entrada: 3.49 2 — Salida correcta: 3.49

Ejemplo de entrada: 3.496 2 — Salida correcta: 3.5

Finalidad: Familiarizarnos con la definición de funciones y las llamadas entre funciones. Dificultad Baja.

12. [Elasticidad precio-demanda] En condiciones normales de mercado, es de esperar que si un producto aumenta de precio, su demanda decaiga. ¿Pero hasta qué punto le puede ser rentable una subida de precio al empresario? Para analizar esta situación, en Economía se usa el concepto de *elasticidad*. Según Wikipedia: La *elasticidad precio de la demanda* mide la sensibilidad (o respuesta) de la cantidad demandada de un bien o servicio ante los cambios en su precio y se define por la siguiente fórmula:

$$EPD = \frac{\% \text{ variación en la cantidad demandada}}{\% \text{ variación en el precio}}$$

Si usamos el concepto de variación porcentual visto en los problemas 5 [Variación porcentual] de la Relación de Problemas I y 22 [Variación porcentual: lectura de varios valores] de la Relación de Problemas II, la definición quedaría así:

$$EPD = \frac{VP_{\text{demanda}}}{VP_{\text{precio}}}$$

Por ejemplo, supongamos que de un determinado producto se venden 10 unidades a 5 euros. Después de subirlo a 6 euros, se venden un total de 4 unidades. El valor de elasticidad sería:

$$EPD = \frac{abs(\frac{4-10}{10})}{abs(\frac{6-5}{5})} = 3$$

Ahora, supongamos que del mismo producto del que se vendían 10 unidades a 5 euros, por el hecho de subirlo a 6 euros se venden un total de 9 unidades. El valor de elasticidad sería:

$$EPD = \frac{abs(\frac{9-10}{10})}{abs(\frac{6-5}{5})} = 0.5$$

En general, valores de elasticidad superiores a 1 indican que no es rentable una subida de precios ya que la demanda, en proporción, caerá mucho más.

Construya un programa que lea el precio inicial, el precio final, la demanda inicial y la demanda final y calcule el valor de elasticidad *EDP* según la fórmula anterior. El programa irá leyendo los cuatro valores anteriores para varios productos. Supondremos que si el usuario introduce un valor positivo en el precio inicial, los otros tres valores también serán positivos. La entrada de datos terminará cuando el usuario introduzca cualquier valor negativo como precio inicial.

Para resolver este problema debe definir las funciones que estime oportunas.

Si lo desea, puede usar el esbozo del programa que puede encontrar en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/IV_ElasticidadPDEsbozo.cpp

Ejemplo de entrada: 5 6 10 4 5 6 10 9 -1

— Salida correcta: Valores de elasticidad precio-demanda: 3 0.5

Ampliación:

El mismo concepto se aplica cuando se produce una bajada de precio. En este caso, es de esperar que una bajada de precio produzca un aumento de la demanda. Valores de elasticidad superiores a 1 indican que es rentable la bajada (porque se venderá mucho más) y menores a 1 que no lo es. Normalmente, si un producto presenta una elasticidad alta para subidas de precio, también tendrá una elasticidad alta para bajadas, es decir, el mercado será muy sensible a pequeñas fluctuaciones de los precios. Un ejemplo son los bienes de lujo. En cambio, los productos de primera necesidad suelen tener una elasticidad baja, aunque claro está, el fabricante no debe subir los precios incontroladamente ya que entran otros factores en juego como por ejemplo la competencia.



Finalidad: Aislar código en funciones que resuelven una tarea concreta. Dificultad Baja.

13. **[Lee entero en rango]** Defina una función `LeeIntRango` para leer un número entero obligando a que esté en un rango `[min, max]`. Para ello, dicha función debe ir leyendo números enteros (de tipo `int`) desde la entrada por defecto, hasta que se introduzca un valor que pertenezca al rango `[min, max]` (no hay ningún límite en el número de intentos). En ese momento, la función terminará su ejecución y devolverá dicho valor.

Escriba un pequeño programa de prueba que lea un número entero obligando a que esté en el intervalo `[0, 100]` (para ello, tendrá que llamar a la función `LeeIntRango`) Imprima el valor devuelto por la función.

Ejemplo de entrada: `-1 200 -50 3` — Salida correcta: `3`

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

14. **[Lee entero mayor o igual que otro]** Defina una función `LeeIntMayorIgual` para leer un entero mayor o igual que un número dado (éste será un parámetro a la función). Para ello, dicha función debe ir leyendo números enteros (de tipo `int`) desde la entrada por defecto, hasta que se lea un valor correcto que sea mayor o igual que el número especificado. La función devolverá dicho valor.

Construya un programa que haga lo siguiente:

- Lea un primer entero `min` sin ninguna restricción.
- Lea un segundo entero `max` obligando a que sea mayor o igual que `min` (debe usar la función `LeeIntMayorIgual`)
- Lea un número entero `n` obligando a que esté en el intervalo `[min, max]`. Para ello, debe usar la función `LeeIntRango` definida en el ejercicio 13 **[Lee entero en rango]**. Imprima en pantalla el último valor (el que pasa el filtro y está dentro del intervalo `[min, max]`).

Ejemplo de entrada:

```
4          min = 4
-1 0 7     Los valores -1 y 0 son "rechazados" -> max = 7
3 -1 8 6   Los valores 3 -1 y 8 son "rechazados"
           al no pertenecer al intervalo [4,7]
```

— Salida correcta: `6`

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

15. [Diseño de funciones sobre cadenas] En este ejercicio no tiene que construir ningún programa. Sólo se le pide que defina las cabeceras de las funciones que se le indique:

- Una función `LeeString` que permita leer caracteres desde el dispositivo de entrada por defecto, hasta que se introduzca un carácter terminador. La función debe devolver un dato de tipo `string` con los caracteres introducidos excepto el terminador. Por ejemplo, si se establece que el terminador es el carácter `@` y el usuario introduce los caracteres `Juan Carlos Cubero@`, la función devolverá la cadena `Juan Carlos Cubero`
- Una función `EliminaUltimos` que, a partir de una cadena de caracteres (`string`), construya una nueva cadena eliminando todos los caracteres **finales** que sean igual a un carácter determinado.

Por ejemplo, al borrar la `T` del final de la cadena `TabcTdTTTT` la función devolvería como resultado la cadena `TabcTd`. Si la cadena fuese `TabcTd`, el resultado de borrar la `T` del final de la cadena sería la misma cadena `TabcTd`, ya que no hay ninguna `T` al final.

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

16. [Uso de funciones sobre cadenas] En este ejercicio vamos a usar las funciones del ejercicio 15 [Diseño de funciones sobre cadenas]. Para ello, puede usar la implementación que se encuentra en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/IV_FuncionesStringEsbozo.cpp

No se preocupe de entender el código que hay dentro de las funciones, ya que únicamente debe saber cómo son su cabeceras. Tenga en cuenta que debe compilar con la opción ISO C++11 tal y como se indica en la página 4 de este guion de prácticas.

Construya un programa que lea caracteres hasta llegar al terminador `#` y construya una variable llamada `original` de tipo `string`. Para ello, debe usar la función `LeeString`. A continuación, el programa leerá un carácter y llamará a la función `EliminaUltimos` para construir una nueva cadena eliminados. Imprima la cadena resultante.

¿Cómo cambiaría la llamada en el `main` a la función `EliminaUltimos` para modificar la misma cadena `original` sobre la que se aplica la función?

Ejemplo de entrada: `TabcTdTTTT#T`

— Salida correcta: `TabcTd`

Ejemplo de entrada: `TabcTd#T`

— Salida correcta: `TabcTd`

Finalidad: Familiarizarnos con la llamada a las funciones y el paso de parámetros. Dificultad Baja.

17. **[DoubleToString]** La función `to_string` de C++11 es muy útil ya que permite convertir un real en una cadena de caracteres. Por ejemplo, `to_string(23.51)` devuelve la cadena "23.510000". Esta función tiene dos inconvenientes:

- a) Siempre fija un redondeo a seis cifras decimales.
- b) No suprime los ceros finales de la cadena de caracteres.

Lo que vamos a hacer en este ejercicio es definir nuestra propia función `DoubleToString` que nos permitirá:

- a) Redondear a cualquier número de cifras decimales (con la restricción de que debe ser menor o igual que 6) Si el número de cifras decimales es mayor o igual que 6, la función redondeará a la sexta cifra decimal.
- b) Eliminaremos los ceros finales.

Para ello, basta hacer lo siguiente:

- Redondear el real usando la función `Redondea` del ejercicio 11 **[Decimal redondeado]**
- Convertir el resultado a `string` usando la función `to_string`
- Eliminar los caracteres finales que sean '0' con la función `EliminaUltimos` del ejercicio 16 **[Uso de funciones sobre cadenas]**
- En el caso de que nos hubiesen pasado un real sin parte decimal, como por ejemplo el 4.0, la cadena resultante no debería ser "4." sino "4". Para quitar ese punto final, puede usar `cadena.pop_back()`; . Lo que hace la anterior sentencia es eliminar el último carácter del string `cadena`.

Construya un programa que lea un real, un número de cifras decimales e imprima la cadena resultante de llamar a la función `DoubleToString`.

Si lo desea, puede usar el esbozo del programa que puede encontrar en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/IV_DoubleToStringEsbozo.cpp

Ejemplo de entrada: 3.49 1 — Salida correcta: "3.5"

Ejemplo de entrada: 3.49 2 — Salida correcta: "3.49"

Ejemplo de entrada: 3.496 2 — Salida correcta: "3.5"

Ejemplo de entrada: 0.12345678 7 — Salida correcta: "0.123457"

Ejemplo de entrada: 3.0 2 — Salida correcta: "3"

Nota. En el ejercicio 49 **[DoubleToString mejorado]** se propone mejorar esta función para que pueda trabajar con cualquier número de cifras decimales (mayor de 6).

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

RELACIÓN DE PROBLEMAS DEL TEMA IV. Funciones y Clases

18. [Tarifa aérea con funciones] Recupere la solución del ejercicio 41 [Tarifa aérea: múltiples billetes] disponible en:

https://decsai.ugr.es/jccubero/FP/II_TarifaAereaMultiplesBilletes.cpp

Re-escribalo usando funciones donde estime oportuno.

Tenga en cuenta que en un problema real, primero debe diseñar las funciones y luego escribir el código. Lo que no debe hacer es escribir un código sin funciones y luego re-escribirlo agrupando partes de código en funciones. En cualquier caso, lo permitimos en estos primeros ejercicios de programación.

Finalidad: Diseñar la solución completa de un programa con funciones. Dificultad Media.

19. [Kaprekar] (*Examen Extraordinario Febrero 2022*) El matemático D.R. Kaprekar descubrió en 1949 una característica del número **6174** (conocido como *constante de Kaprekar*), la cual está relacionada con un proceso iterativo. Dado un número estrictamente positivo de cuatro cifras (o menos) que tenga al menos dos dígitos diferentes (los números con menos de cuatro dígitos se completan colocando el dígito 0 al principio, por lo que el número 9 se convierte en 0009, por ejemplo), el proceso iterativo es el siguiente:

- Reordenar los dígitos de forma ascendente y descendente para así formar dos nuevos números.
- Restar el menor al mayor.
- Volver al paso primero considerando ahora el resultado de la resta del paso segundo.

La curiosa característica de este proceso es que siempre converge al número **6174** (y como mucho, en **7** iteraciones).

Observe cómo se alcanza la constante de Kaprekar para los números 3524, 25 y 1121 en 7 ó menos iteraciones:

	3524		25		1121
Iteración 1	5432 - 2345 = 3087		5200 - 0025 = 5175		2111 - 1112 = 0999
Iteración 2	8730 - 0378 = 8352		7551 - 1557 = 5994		9990 - 0999 = 8991
Iteración 3	8532 - 2358 = 6174		9954 - 4599 = 5355		9981 - 1899 = 8082
Iteración 4			5553 - 3555 = 1998		8820 - 0288 = 8532
Iteración 5			9981 - 1899 = 8082		8532 - 2358 = 6174
Iteración 6			8820 - 0288 = 8532		
Iteración 7			8532 - 2358 = 6174		

Los únicos números de cuatro cifras para los que el proceso de Kaprekar no converge a 6174 (no se puede aplicar) son aquellos que tienen todas las cifras iguales (1111, 2222, 3333, ...).

Teniendo en cuenta lo anterior, se pide construir un programa **main** que lea un valor **min** y otro valor **max**, ambos con un máximo de 4 dígitos y muestre, para cada uno de ellos, si no se puede aplicar Kaprekar (porque tiene todas sus cifras iguales) y en caso contrario cuántas iteraciones tarda en alcanzar el número **6174**. Para ello, dispone de las siguientes funciones que ya están implementadas en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/IV_KaprekarEsbozo.cpp

- `string ToStringInt`
(`int entero`, `int min_num_casillas`, `char relleno`)
devuelve el valor textual (en forma de `string`) del dato `entero`. Para ello emplea un mínimo de `min_num_casillas` casillas. Si `min_num_casillas` es mayor que el que se necesita, se rellena, al principio, con el carácter `relleno`. Por ejemplo, `ToStringInt(153, 4, '0')` devuelve "0153", mientras que `ToStringInt(153, 1, '0')` devuelve "153".
- `bool TodosIguales(string cadena)` indica si los caracteres de la cadena son todos iguales o no. Por ejemplo, `TodosIguales("1111")` devuelve `true`, mientras que `TodosIguales("1211")` devuelve `false`.
- También puede usar la función `int stoi(string cadena)` de la biblioteca `string`. Esta función convierte el parámetro `cadena` al entero correspondiente. Por ejemplo, `stoi("0015")` devuelve 15 y `stoi("1234")` devuelve 1234.

Para resolver este problema, se le pide que defina las siguientes funciones:

- Una función que cree un `string` reordenando los caracteres de otro `string`. La usará para obtener el valor menor en el proceso de Kaprekar.
- Una función que cree un `string` invirtiendo los caracteres de otro `string`. La usará para obtener el valor mayor, a partir del valor menor.

Puede definir las **funciones** adicionales que estime oportuno.

Ejemplo de entrada: 9 12

— Salida correcta:

```
9 -> Todas las cifras son iguales
10 -> Converge en 5 iteraciones
11 -> Todas las cifras son iguales
12 -> Converge en 3 iteraciones
```

Finalidad: Diseñar la solución completa de un programa con funciones. Dificultad Media.

20. [Errores en funciones void] Encuentre los errores, si los hubiese, en las siguientes funciones void:

```
void Imprime(double valor){
    double valor;

    cout << valor;
}
void Cuadrado (int entero){
    return entero*entero;
}
int Cuadrado (int entero){
    entero = entero*entero;
}
```

Finalidad: Familiarizarnos con los void, el paso de parámetros y el ámbito de las variables. Dificultad Baja.

21. [Mensaje inicial] Queremos mostrar al inicio de nuestros programas un mensaje inicial (de tipo string) enmarcado en sendas líneas de caracteres, como por ejemplo:

```
-----
Elimina Caracteres
-----
```

o bien:

```
*****
Cálculo del IVA
*****
```

Observe que queremos poder usar cualquier carácter para enmarcar. Además, el número de caracteres usado para enmarcar se ha de ajustar automáticamente al tamaño del texto del mensaje.

Se pide construir las funciones que estime oportuno para realizar esta tarea.

Construya un programa de prueba que lea desde teclado el carácter que se usará para enmarcar. A continuación lea un dato de tipo `string` con la función `LeeString` vista en el ejercicio 15 [Diseño de funciones sobre cadenas] (use como terminador el carácter `@`).

Si lo desea, puede usar el esbozo del programa que puede encontrar en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/IV_MensajeEsbozo.cpp

El programa debe mostrar en pantalla la cadena enmarcada con dicho carácter. Necesitará usar los siguientes recursos sobre una variable cadena de tipo `string`:

RELACIÓN DE PROBLEMAS DEL TEMA IV. Funciones y Clases

- La expresión `cadena.size()` devuelve un entero con la longitud actual de la variable `cadena`
- La sentencia `cadena.push_back(caracter);` añade `caracter` (de tipo `char`) al final de `cadena`
- La expresión `cadena + otra_cadena` devuelve un nuevo `string` con la concatenación de ambas cadenas.

Ejemplo de entrada: `-Bienvenidos@`

— Salida correcta:

```
-----  
Bienvenidos  
-----
```

Ejemplo de entrada: `*Fundamentos de Programación@`

— Salida correcta:

```
*****  
Fundamentos de Programación  
*****
```

Finalidad: Trabajar con funciones void. Dificultad Baja.

Problemas sobre Clases

22. [Recta con datos miembro públicos] En este programa vamos a trabajar con datos miembro públicos. Posteriormente se modificará para trabajar con datos miembro privados.

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

$$Ax + By + C = 0$$

Si lo desea, puede usar el recurso disponible en el siguiente enlace para ver la representación gráfica de una recta en función de los coeficientes, la pendiente, etc.:

<https://www.geogebra.org/m/CCPQjtDc>

Defina la clase `Recta`. En este ejercicio utilice únicamente datos miembro públicos. Éstos serán los tres coeficientes de la recta. Añada los siguientes métodos:

- Método `Pendiente` para obtener la **pendiente** de la recta, aplicando la fórmula:

$$\text{pendiente} = -A / B$$

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

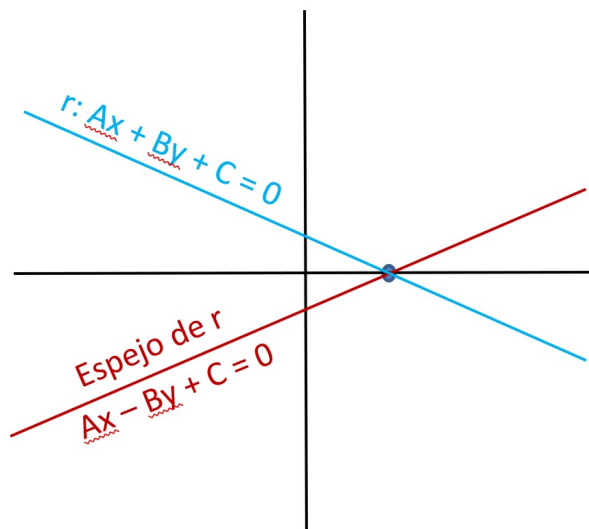
- Método `Ordenada` para obtener la **ordenada** (y) dado un valor de abscisa x , aplicando la fórmula:

$$y = (-C - xA) / B$$

- Método `Abscisa` para obtener la **abscisa** (x) dado un valor de ordenada y , aplicando la fórmula:

$$x = (-C - yB) / A$$

- Método `EspejoAbsc` para transformar la recta en otra distinta que pase por el mismo punto de intersección con el eje de las abscisas y se obtenga como un **espejo** de la recta original (da igual el eje con respecto al que se hace el espejo pues la recta resultante es la misma) Para ello, basta modificar el valor del coeficiente B cambiándolo de signo. Observe que no se le pide que construya una recta nueva sino que modifique la propia recta sobre la que se aplica el método.



Nota: En principio, tendríamos que tener en cuenta dos casos especiales:

- En el caso de que B fuese cero, la recta es vertical por lo que la recta espejo es la misma.
- En el caso de que A fuese cero, la recta es horizontal y no corta al eje de las abscisas, por lo que no tiene sentido calcular la recta espejo.

Por lo tanto, no hay que programar nada para tener en cuenta estos casos especiales ya que, en ambos casos, no hay que modificar ningún coeficiente de la recta.

- Método `AnguloRadEjeHoriz` para obtener el **ángulo** θ de la recta con el eje horizontal, aplicando la fórmula:

$$\theta = \arctan(\text{pendiente de la recta})$$

Observe que `arctan` denota el arco tangente: está implementada en la función `atan` de `cmath`. Para enfatizar que el valor devuelto no son grados sexagesimales sino radianes, hemos incluido las letras `Rad` en el nombre de la función `AnguloRadEjeHoriz`.

Cree un programa principal que haga lo siguiente:

- Cree dos objetos `recta1` y `recta2` de la clase `Recta`.
- Lea seis reales desde teclado. Puede asumir que los valores se introducirán correctamente y por tanto A y B no serán simultáneamente cero. Asigne los tres primeros a los coeficientes de `recta1` y los otros tres a `recta2`. Imprima en pantalla el término independiente de cada recta.
- Calcule e imprima la pendiente de cada recta.
- Calcule e imprima el ángulo en radianes con respecto al eje horizontal de cada recta.

- Lea un valor de abscisa e imprima la ordenada según `recta1`. A continuación lea un valor de ordenada e imprima la abscisa que le corresponde.
- Ejecute el método `EspejoAbsc` sobre `recta1` e imprima en pantalla la pendiente nueva (será la misma que antes pero cambiada de signo).

Si lo desea, puede usar el esbozo del programa que puede encontrar en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/IV_RectaEsbozo.cpp

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

— Salida correcta:

```
Término independiente de la recta 1: -1
Término independiente de la recta 2: 9
Pendiente de la recta 1: -1.5
Pendiente de la recta 2: 0.875
Ángulo en radianes con el eje horizontal de la recta 1: -0.982794
Ángulo en radianes con el eje horizontal de la recta 2: 0.71883
Ordenada de la recta 1 en 5 = -7
Abscisa de la recta 1 en 6 = -3.66667
Pendiente del espejo de recta 1: 1.5
```

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

23. **[Dinero con datos miembro públicos]** En este programa vamos a trabajar con datos miembro públicos. Posteriormente se modificará para trabajar con datos miembro privados.

Como ya se ha visto en los apuntes de clase, el tipo de dato `double` no es adecuado para representar cantidades monetarias debido a los problemas de precisión y redondeo. Para resolver este problema, queremos construir nuestro propio tipo de dato definiendo la clase `Dinero`. Obviamente, la funcionalidad de la clase será limitada ya que no deja de ser un ejemplo docente. En cualquier caso, en un ejemplo real, tendremos que utilizar una clase similar (busque en Internet, por ejemplo, `Binary Coded Decimal`)

La clase `Dinero` tendrá dos datos miembro, euros y centimos. Debe definir el método `SetEurCent` para asignar una cantidad dada de euros y céntimos. Por ejemplo, si desde el programa principal ejecutamos la sentencia:

```
un_dinero.SetEurCent(20, 115);
```

el método `SetEurCent` debe hacer la conversión pertinente para guardar 21 en el dato miembro euros y 15 en el dato miembro centimos (115 céntimos son 1 euro y 15 céntimos)

Construya un programa que lea el precio de dos productos (euros y céntimos de un producto y euros y céntimos del otro) y construya dos objetos de la clase `Dinero` para almacenar el precio de cada producto. A continuación, el programa principal debe recuperar las cantidades de ambos precios (accediendo a los datos miembros públicos de cada objeto), sumarlos (la suma de las cantidades enteras la tiene que hacer en el propio `main`) y construir un tercer objeto con el resultado de la suma. Finalmente, el programa debe imprimir en pantalla el número de euros y céntimos del tercer objeto.

Ejemplo de entrada: 20 190 30 115

— Salida correcta: La suma es 53 euros y 5 céntimos

Nota. Tal vez habrá pensado que sería conveniente definir dentro de la propia clase `Dinero` el método `Suma` para poder sumar dos objetos de la clase `Dinero`. En la Relación de Problemas V verá cómo hacerlo.

Si lo desea, puede usar el esbozo del programa que puede encontrar en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/IV_DineroEsbozo.cpp

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

24. [Recta con datos miembro privados] Recupere la solución del ejercicio 22 [Recta con datos miembro públicos]. Cambie ahora los datos miembro públicos y póngalos *privados*. Tendrá que añadir métodos para asignar y ver los valores de los datos miembro. Añada métodos para asignar un valor a cada uno de los tres datos miembro. Modifique el `main` para tener en cuenta estos cambios.

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

IMPORTANT

Cambie ahora la *política de acceso a los datos miembros*: en vez de usar un método para asignar un valor a cada dato miembro, defina un único método `SetCoeficientes` para asignar los tres a la misma vez.

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

Si lo desea, puede usar el esbozo del programa que puede encontrar en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/IV_RectaEsbozo.cpp

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

25. **[Cronómetro]** Considere la siguiente definición de la clase Cronometro:

```
class Cronometro{
private:
    ...
public:
    void Reset(){
        ...
    }
    long long NanoSegundosTranscurridos(){
        ...
    }
};
```

Esta clase sirve para medir el tiempo de ejecución de un conjunto de instrucciones. Para usarla, haga copy-paste del código que se puede encontrar en el siguiente enlace (no olvide insertar también los `#include` que aparecen al inicio.):

https://decsai.ugr.es/jccubero/FP/IV_ClaseCronometro.cpp

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

Se pide añadir un nuevo método `MiliSegundosTranscurridos` para saber cuántos milisegundos han transcurrido. Este método debe llamar al anterior.

Recupere alguna de las soluciones de los ejercicios de la Relación de Problemas III y mida los tiempos de ejecución. Por ejemplo, realice 10 millones de iteraciones en el bucle que calcula el valor de π según se vio en el ejercicio 32 **[Aproximación de π por Madhava sin usar pow]**

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

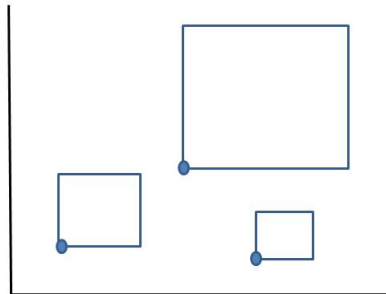
26. **[Dinero con datos miembro privados]** Modifique la solución del ejercicio 23 **[Dinero con datos miembro públicos]** de esta Relación de Problemas y cambie el ámbito de los datos miembro a `private`. Por lo tanto, tendrá que proporcionar métodos `Get` para consultar los euros y los céntimos de cada objeto y modificar adecuadamente el programa principal.

Si lo desea, puede usar el esbozo del programa que puede encontrar en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/IV_DineroEsbozo.cpp

Finalidad: Trabajar con los datos miembro privados. Dificultad Baja.

27. [Cuadrado] Se desea implementar una clase Cuadrado para representar figuras geométricas de tipo cuadrado. Un cuadrado viene definido por la coordenada de la esquina inferior izquierda (x, y) y la longitud del lado. Por lo tanto, el tipo de cuadrados que se consideran son los que su base es paralela al eje de las abscisas.



Defina una clase en C++ para representar un cuadrado. Tendrá que definir los datos miembro *privados* y los métodos Get para devolver su valor. Defina también el/los métodos Set para asignarles un valor. Tendrá que decidir cuántos métodos Set define dependiendo de si quiere asignar valores a los datos miembro todos a la misma vez o por separado.

Debe definir métodos para:

- Calcular el área del cuadrado.
¿Añadimos *area* como dato miembro del cuadrado?
- Calcular el perímetro del cuadrado.
¿Añadimos *perimetro* como dato miembro del cuadrado?

Construya un programa principal que lea los datos de dos cuadrados. Debe leer las dos coordenadas de la esquina, así como la longitud del primer cuadrado y asignar dichos valores a los datos miembro de una variable *parcela* de tipo Cuadrado. Haga lo mismo con un segundo objeto *otra_parcela* de la clase Cuadrado.

Calcule el área y perímetro de ambos cuadrados e imprímalos en pantalla. Imprima también las coordenadas de la esquina inferior izquierda, accediendo a los correspondientes métodos Get del cuadrado.

Si lo desea, puede usar el esbozo del programa que puede encontrar en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/IV_CuadradoEsbozo.cpp

Ejemplo de entrada:

3.4 5.7 2.9
-5.6 -4.1 1.8

— Salida correcta:

Coordenadas: 3.4 , 5.7
Longitud: 2.9
Área: 8.41
Perímetro: 11.6

Coordenadas: -5.6 -4.1
Longitud: 1.8
Área: 3.24
Perímetro: 7.2

Nota. Aunque no se le pida que lo haga, habrá situaciones en las que podría interesar suprimir los métodos que modifican los datos miembro. De esta forma, una vez creado el objeto (pasándole los datos apropiados en el constructor) ya no podríamos modificar los datos miembro. Esto es útil en aquellas situaciones en las que no queremos permitir que el estado del objeto cambie, una vez que se ha creado.

Finalidad: Definir una clase sencilla. Dificultad Baja.

28. [Cuadrado con constructor] Recupere la solución del ejercicio 27 [Cuadrado] y añada un constructor al cuadrado para que el objeto esté en un estado válido en el mismo momento de su definición. El constructor deberá tener como parámetros, obligatoriamente, las coordenadas de la esquina y la longitud del lado. Tendrá que modificar el `main` para tener en cuenta este cambio.

Finalidad: Definir un constructor. Dificultad Baja.

29. [Recta con constructor] Recupere la solución del ejercicio 24 [Recta con datos miembro privados] que puede encontrar en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/IV_RectaPrivado.cpp

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

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

Finalidad: Familiarizarnos con la definición de constructores y métodos privados. Dificultad Baja.

30. [Generador aleatorio] Vamos a trabajar con una clase `GeneradorAleatorioEnteros` que me va a permitir generar secuencias de números aleatorios enteros en un rango de valores. El código de dicha clase lo puede encontrar en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/IV_GeneradorAleatorioEsbozo.cpp

GeneradorAleatorioEnteros	
+	<code>GeneradorAleatorioEnteros()</code>
+	<code>GeneradorAleatorioEnteros(int min, int max)</code>
+	<code>int Siguiente()</code>

El objetivo de este ejercicio es que aprenda a utilizar sus métodos sin que necesite entender cómo están implementados. Por lo tanto, basta con que entienda las cabeceras de los constructores y del método `Siguiente`. Veámoslo.

Esta clase tiene dos constructores.

- a) Uno de los constructores no tiene parámetros. Lo usaremos cuando queramos generar únicamente ceros y unos, por lo que construiríamos el objeto en la siguiente forma:

```
// Constructor sin parámetros
GeneradorAleatorioEnteros aleat_0_1;
```

- b) El otro constructor tiene dos parámetros, `min` y `max` que delimitan el rango correspondiente de valores aleatorios a generar. Por ejemplo, si quisiéramos un objeto para generar números de la lotería primitiva (entre 1 y 49) tendríamos que poner lo siguiente:

```
// Constructor con parámetros
GeneradorAleatorioEnteros aleat_loteria(1, 49);
```

Cada vez que se llame al método `Siguiente`, éste devolverá un valor aleatorio en el rango especificado. Para generar, por ejemplo, los 6 números de la primitiva haríamos lo siguiente:

```
for (int i = 0; i < 6; i++)
    cout << aleat_loteria.Siguiente();
```

Utilice esta clase para hacer lo siguiente: En primer lugar debe generar un número aleatorio entre 1 y 5 (llámelo `num_valores_a_generar`) Muéstrelo por pantalla. A continuación debe generar tantos números aleatorios entre 0 y 1 como indique el anterior valor `num_valores_a_generar`. Muestre por pantalla los valores generados.

Haga lo anterior 4 veces.

Por ejemplo, una posible salida sería:

RELACIÓN DE PROBLEMAS DEL TEMA IV. Funciones y Clases

```
2    0 1          -> num_valores_a_generar = 2
5    0 0 1 0 1    -> num_valores_a_generar = 5
3    1 0 0        -> num_valores_a_generar = 3
2    1 1          -> num_valores_a_generar = 2
```

No olvide insertar en su programa los `#include` de las bibliotecas `random` y `chrono` que aparecen al inicio del fichero de esbozo indicado anteriormente .

Nota:

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

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

31. [Diseño de la interfaz de la clase `SecuenciaCaracteres`] En este ejercicio se pide que defina las cabeceras de varios métodos. No debe implementar ninguno de los métodos. Sólo se pide que escriba las cabeceras y cómo sería la llamada a dichos métodos desde otro sitio (por ejemplo, desde el `main`) pero no tiene que construir un programa completo que compile.

Se quiere definir una clase `SecuenciaCaracteres` similar a la clase `string` para manipular secuencias de caracteres (de datos de tipo `char`). Por ejemplo, en un determinado momento, podríamos tener almacenada la siguiente secuencia:

Juan Carlos

Como dato miembro privado, se recomienda usar un vector de caracteres. Ya sabemos que debemos reservar memoria suficiente. Supondremos que el máximo será 100. En vez de usar el literal 100, mejor usamos una constante. En ese caso, C++ obliga a que sea una *constante estática* (si aún no ha visto las constantes estáticas en clase de Teoría, no se preocupe ya que no se le pide que el programa compile). Nos quedaría lo siguiente:

```
private:
    static const int TAM = 100;
    char caracteres[TAM];
```

Tenemos dos alternativas para marcar el bloque del vector que esté siendo utilizado en cada momento:

- a) O bien usamos un terminador fijo, por ejemplo, `#`, al final del bloque usado. Por ejemplo, en un momento dado, el vector `caracteres` contendría:

```
'J' 'u' 'a' 'n' ' ' 'C' 'a' 'r' 'l' 'o' 's' '#' ' ' ? ? .... ?
```

- b) O bien usamos un dato miembro privado `utilizados` que nos diga cuántos elementos estamos usando en cada momento. Por ejemplo, en un momento dado, el vector `caracteres` contendría:

`'J' 'u' 'a' 'n' ' ' 'C' 'a' 'r' 'l' 'o' 's' ? ? ? ?`

y el dato `utilizados` valdría 11.

En este ejercicio, elegimos la segunda forma ya que no requiere *reservar* un carácter especial como terminador. En cualquier caso, en otras aplicaciones podría ser conveniente usar la primera forma.

¿Qué métodos definiría para manipular la secuencia? Al menos debe definir las cabeceras de los métodos para realizar lo siguiente:

- a) Método `Aniade` para añadir un carácter al final de la secuencia.

Tenga en cuenta que a una variable `cadena` de tipo `string` se le puede asignar directamente `cadena = "Hola"`. Sin embargo, eso no lo sabemos hacer por ahora con objetos de nuestras propias clases. Por lo tanto, la única forma de añadir caracteres a un objeto de la clase `SecuenciaCaracteres` sería hacerlo de uno en uno. Así pues, llamando a un método `Aniade`, añadiríamos la `'H'`, luego la `'o'` y así sucesivamente.

- b) Método `Utilizados` para obtener la longitud actual de la secuencia (el número de caracteres que contiene en ese momento)

- c) Método `Invierte` para invertir la secuencia. En el ejemplo anterior, la secuencia se quedaría en:

`solraC nauJ`

- d) Método `PrimeraOcurrencia` para buscar la primera ocurrencia de un carácter.

- e) Método `EliminaOcurrencias` para eliminar las ocurrencias de un carácter. Por ejemplo, después de eliminar el carácter `' '`, la secuencia quedaría así:

`JuanCarlos`

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

32. **[Diseño de la interfaz de la clase `Instante`]** En este ejercicio se pide que defina las cabeceras de varios métodos. No debe implementar ninguno de los métodos. Sólo se pide que escriba las cabeceras y cómo sería la llamada a dichos métodos desde otro sitio (por ejemplo, desde el `main`) pero no tiene que construir un programa completo que compile.

Se quiere definir una clase `Instante` para representar un instante de tiempo dentro de un día. Por lo tanto, la clase debe representar un número de horas, minutos y segundos.

¿Qué datos miembro debería tener? Defina las cabeceras de métodos para realizar las siguientes tareas:

- Obtener el número de segundos y minutos transcurridos desde las 0h 0min 0seg. Por ejemplo, si el instante es 1h 2min 5seg, el número total de segundos transcurridos es de 3725 y el de minutos 62.
- Establecer el instante a partir del número de segundos transcurridos. Por ejemplo, si se establece a 3725, el instante debe contener 1h 2min 5seg.
- Puede especificar las cabeceras de todos los métodos adicionales que estime oportuno.

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

33. [Diseño de la interfaz de la clase `FormateadorDoubles`] En este ejercicio se pide que defina las cabeceras de varios métodos. *No debe implementar* ninguno de los métodos. Sólo se pide que escriba las cabeceras y cómo sería la llamada a dichos métodos desde otro sitio (por ejemplo, desde el `main`) pero no tiene que construir un programa completo que compile.

En el ejercicio 17 [DoubleToString] de esta Relación de Problemas habíamos transformado un real a una cadena de caracteres (redondeando previamente el real a cierto número de cifras decimales) Vamos a extender esta funcionalidad.

Dado un dato de tipo `double`, queremos *formatearlo* adecuadamente y construir un dato de tipo `string`, especificando lo siguiente:

- Un delimitador a la izquierda y otro a la derecha.
- Si el separador es el punto o la coma.
- Número de decimales a obtener.

Por ejemplo, si el delimitador izquierda se fija a la cadena "<<", el derecha a ">>", el separador de decimales a la coma y el número de decimales a 7, el resultado de formatear el número real 0.12345678 sería la cadena "<<0,1234568>>"

Si quisiéramos realizar la conversión con una función, su cabecera podría ser la siguiente:

```
string DoubleToString (double real,
                        string delim_izda, string delim_dcha,
                        char caracter_separador,
                        int num_decim_redond)
```

Una llamada de ejemplo sería la siguiente:

```
cadena = DoubleToString(0.12345678, "<<", ">>", '.', 7);
cadena = DoubleToString(37.5467891, "<<", ">>", '.', 7);
```

Lo que se pide en este ejercicio es resolver el mismo problema pero diseñando una clase `FormateadorDoubles`. El objetivo es que la conversión se realice a través de un método `GetCadena` de la siguiente forma:

```
int main(){
    FormateadorDoubles formateador(...);
    ...
    cadena = formateador.GetCadena(0.12345678);
    cadena = formateador.GetCadena(37.5467891);
}
```

Debe decidir qué parámetros pasamos en el constructor y qué métodos públicos tendría la clase.

En el diseño de esta clase debe decidir:

- ¿Algún dato miembro tiene un valor por defecto?
- ¿Cómo se especificarán los delimitadores (izquierda y derecha) y el separador (punto o coma)?
- ¿Cuántos constructores va a necesitar?, etc.

Incluya el código de cómo sería un programa principal de prueba. Recuerde que no se le pide implementar la clase, sólo que diseñe las cabeceras de los métodos.

Finalidad: Diseño de una clase. Dificultad Media.

34. **[Diseño de la interfaz de la clase SimuladorDeposito]** En este ejercicio se pide que defina las cabeceras de varios métodos. No debe implementar ninguno de los métodos. Sólo se pide que escriba las cabeceras y cómo sería la llamada a dichos métodos desde otro sitio (por ejemplo, desde el `main`) pero no tiene que construir un programa completo que compile.

Se quiere definir la clase `SimuladorDeposito` para simular préstamos, ofreciendo la funcionalidad descrita en los ejercicios 24 **[Interés bancario (capital reinvertido)]** y 25 **[Interés bancario (doblar)]** de la relación de problemas II (página Problemas-34).

Por tanto, la clase debe proporcionar, para un capital y unos intereses dados, métodos para:

- a) Calcular el capital que se obtendrá al cabo de un número de años,
- b) Calcular el número de años que deben pasar hasta obtener el doble de la cantidad inicial.

A la hora de diseñar la clase, tendremos que analizar cuestiones como:

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

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

35. [SimuladorDeposito (implementación)] Implemente la clase indicada en el ejercicio 34 [Diseño de la interfaz de la clase SimuladorDeposito] de esta Relación de Problemas. Construya un programa principal de prueba.

Finalidad: Trabajar con una clase. Dificultad Baja.

36. [Instante (implementación)] Sobre la solución del ejercicio 32 [Diseño de la interfaz de la clase Instante] implemente por completo la clase, escribiendo el código de los métodos.

Puede re-utilizar la solución del ejercicio 13 [Segundos entre dos instantes] de la Relación de Problemas I disponible en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/I_DiferenciaInstantes.cpp

Finalidad: Trabajar con una clase. Dificultad Baja.

37. [Instante (uso)] Recupere la solución del ejercicio 36 [Instante (implementación)] . Cree un programa que lea un número de horas, minutos y segundos iniciales/finales. El programa creará los objetos `instante_inial` e `instante_final` y calculará el número de segundos que hay de diferencia entre ambos instantes. A continuación creará un objeto `instante_diferencia` correspondiente a la cantidad de segundos de diferencia entre el instante final y el inicial. Por ejemplo, si el número de segundos entre los instantes inicial y final es 67 segundos, el objeto `instante_diferencia` deberá contener el instante 0 hor, 1 min, 7 seg. Imprima en pantalla el valor de hora, minuto y segundo de `instante_diferencia`. El programa también imprimirá el total de segundos y minutos enteros que hay en `instante_diferencia`.

Si lo desea, puede usar el esbozo del programa que puede encontrar en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/IV_InstanteUsoEsbozo.cpp

Ejemplo de entrada: 2 10 4 3 12 5

— Salida correcta:

Diferencia: 1h 2' 1''

Total segundos diferencia: 3721

Total minutos diferencia: 62

Finalidad: Diseño de una clase. Dificultad Baja.

38. [FormateadorDoubles (implementación)] Defina los métodos de la clase `FormateadorDoubles` indicada en el ejercicio 33 [Diseño de la interfaz de la clase `FormateadorDoubles`] .

39. [FormateadorDoubles (uso)] Recupere la solución del ejercicio 38 [FormateadorDoubles (implementación)] disponible en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/IV_FormateadorDoubles.cpp

Construya un programa para que lea un número real de tipo `double` y construya varias cadenas de tipo `string` a partir de él, usando la clase `FormateadorDoubles`. En primer lugar, debe usar el constructor por defecto de la clase `FormateadorDoubles`, y mostrar en pantalla el resultado de llamar al método `GetCadena`. A continuación, modifique el separador para que sea la coma, establezca a 3 el número de decimales y establezca los delimitadores izquierda y derecha a "<<" y ">>" respectivamente. Imprima la cadena resultante.

Ejemplo de entrada: 3.5679 — Salida correcta: 3.57 <<3,568>>

Finalidad: Uso de una clase con constructores. Dificultad Baja.

40. [Mayoría absoluta] Vamos a extender el ejercicio 11 [Mayoría absoluta] de la Relación de Problemas II.

Se quiere construir un programa que calcule las posibles coaliciones de dos partidos que se pueden realizar para alcanzar la mayoría absoluta después de una votación en la que participan varios partidos.

El programa leerá el número de votos presenciales que ha obtenido un partido y a continuación leerá el número de votos por correo obtenidos por el mismo partido. A continuación se leerán los mismos datos de otros partidos. La introducción de datos parará cuando se introduzca un -1.

El programa debe mostrar si hay algún partido que ha alcanzado la mayoría absoluta, es decir, que el total de votos obtenidos (los presenciales más los obtenidos por correo) sea mayor estricto que la mitad de la suma total de votos de todos los partidos. En el caso de que ningún partido haya alcanzado la mayoría absoluta, el programa debe mostrar las coaliciones de dos partidos que lleven a una mayoría absoluta. Debe tener en cuenta que puede haber más de una posible coalición que obtenga dicha mayoría absoluta.

Los partidos vendrán identificados por un entero empezando desde 0 (asignado automáticamente por el programa según se van introduciendo los datos)

Debe usar un vector `coaliciones` en el que cada componente sea un `struct` que contenga dos campos con los índices de los partidos de cada coalición.

Si se desea, puede usar el esqueleto del programa disponible en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/IV_MayoriaAbsolutaEsbozo.cpp

Por simplificar, se utilizarán enteros pequeños en los casos de prueba:

Ejemplo de entrada: 2 4 3 40 1 5 2 6 -1

— Salida correcta: El partido con índice 1 alcanzó la mayoría absoluta

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

— Salida correcta: Ningún partido alcanzó la mayoría absoluta

Posibles coaliciones 1 3, 2 3:

Finalidad: Trabajar con un vector de structs. Dificultad Baja.

Problemas sobre vectores dentro de clases

En los ejercicios que pida trabajar sobre la clase `SecuenciaCaracteres`, use la siguiente definición:

SecuenciaCaracteres	
- const int	TAMANIO
- char	v[TAMANIO]
- int	util
- void	IntercambiaComponentes(int pos_izda, int pos_dcha)
+	SecuenciaCaracteres()
+ int	Utilizados()
+ int	Capacidad()
+ void	Aniade(char nuevo)
+ void	Modifica(int pos_a_modificar, char valor_nuevo)
+ char	Elemento(int indice)
+ void	EliminaTodos()
+ int	PrimeraOcurrenciaEntre(int pos_izda, int pos_dcha, char buscado)
+ int	PrimeraOcurrencia(char buscado)
+ int	BusquedaBinaria(char buscado)
+ int	PosMinimoEntre(int izda, int dcha)
+ int	PosMinimo()
+ void	Inserta(int pos_insercion, char nuevo)
+ void	Elimina(int pos_a_eliminar)
+ void	Ordena_por_Seleccion()
+ void	Ordena_por_Insercion()
+ void	Ordena_por_Burbuja()
+ void	Ordena_por_BurbujaMejorado()
+ string	ToString()

Puede encontrar el código en la dirección siguiente:

https://decsai.ugr.es/jccubero/FP/IV_SecuenciaCaracteres.cpp

41. **[Clase `SecuenciaCaracteres`: Palíndromo e Invierte]** Trabaje sobre la clase `SecuenciaCaracteres` tal y como viene descrita arriba. Añádale métodos para realizar las siguientes tareas (descritas en el ejercicio 2 **[Palíndromo e Invierte]** de la Relación de Problemas III)
- `EsPalindromo` para comprobar si la secuencia es o no un palíndromo.
 - `Invierte` para invertir la secuencia. Este método modificará la secuencia sobre la que se aplica. Para implementarlo, desde dentro del método, debe llamar a `IntercambiaComponentes`

- Supongamos que decidimos cambiar el método `IntercambiaComponentes` y hacer que sea público en vez de privado. ¿Añadiría o quitaría algo en la implementación del método?

La solución de estas tareas utilizando un vector definido en el programa principal puede encontrarse en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/III_PalindromoInvierte.cpp

Incluya un programa principal de prueba similar al del ejercicio 2 [Palíndromo e Invierte] de la Relación de Problemas III. Así pues, el programa debe ir leyendo caracteres e introduciéndolos en un objeto de la clase `SecuenciaCaracteres`. Use como terminador el carácter `#`. Una vez hecho esto, llame al método `Palindromo`. Si la secuencia es un palíndromo se mostrará en pantalla el mensaje "Es un palíndromo". En caso contrario debe llamar al método `Invierte` y mostrar en pantalla la secuencia invertida.

Puede usar el esbozo del programa que se encuentra en esta dirección:

https://decsai.ugr.es/jccubero/FP/IV_SecCaract_PalindromoInvierteEsbozo.cpp

Ejemplo de entrada: `a#` — Salida correcta: `Es un palíndromo`

Ejemplo de entrada: `abcba#` — Salida correcta: `Es un palíndromo`

Ejemplo de entrada: `abccba#` — Salida correcta: `Es un palíndromo`

Ejemplo de entrada: `abcdba#`

— Salida correcta: `No es un palíndromo. Secuencia invertida: abdcba`

Finalidad: Trabajar con un vector dentro de una clase. Dificultad Baja.

42. [Clase `SecuenciaCaracteres`: Elimina ocurrencias ineficiente] En el ejercicio 16 [Elimina ocurrencias de una componente -versión ineficiente-] de la Relación de Problemas III se vio cómo eliminar todas las apariciones que hubiese de un determinado carácter `a_borrar`, dentro de un vector de caracteres.

Por ejemplo, después de eliminar el carácter `'o'` del vector

```
{ 'S', 'o', 'Y', ' ', 'y', 'o' }
```

éste debe quedarse con:

```
{ 'S', 'Y', ' ', 'y' }.
```

Un algoritmo (muy ineficiente) que resolvía este problema era el siguiente:

Recorrer todas las componentes del vector

Si la componente es igual al carácter `a_borrar`, eliminarla
(desplazando hacia la izda las componentes que hay a su dcha)

Una posible implementación del anterior algoritmo en un programa que trabaja directamente con un vector declarado dentro del `main` se encuentra en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/III_EliminaOcurrenciasIneficiente.cpp

Lo que queremos hacer ahora es implementar esta funcionalidad dentro de la clase `SecuenciaCaracteres`, cuyo código base se encuentra en:

https://decsai.ugr.es/jccubero/FP/IV_SecuenciaCaracteres.cpp

Por lo tanto, se pide que añada el método `EliminaOcurrencias` para eliminar todas las apariciones de un determinado carácter `a_borrar`. El método debe modificar la secuencia sobre la que se aplica.

Hágalo implementando el algoritmo anterior, aunque sea ineficiente. Para ello, dentro del método `EliminaOcurrencias`, debe llamar dentro de un bucle al método `Elimina` (que borra un único carácter). La implementación del método `Elimina` se encuentra en el código base de la clase `SecuenciaCaracteres` proporcionado anteriormente.

Incluya un programa principal de prueba similar al del ejercicio 16 [Elimina ocurrencias de una componente -versión ineficiente-] de la Relación de Problemas III, de forma que los caracteres que se vayan leyendo hasta llegar al terminador `#` se van añadiendo al objeto de la clase `SecuenciaCaracteres`. El programa leerá a continuación el carácter a eliminar y llamará al método `EliminaOcurrencias`, imprimiendo en pantalla el resultado.

Puede usar el esbozo del programa que se encuentra en esta dirección:

https://decsai.ugr.es/jccubero/FP/IV_SecCaract_EliminaOcurrIneficienteEsbozo.cpp

Ejemplo de entrada: `maaaovaiala#a`

— Salida correcta: `movil`

Ejemplo de entrada: `aaaaa#a`

— Salida correcta: *secuencia vacía*

Finalidad: Trabajar con un vector dentro de una clase. Llamadas entre métodos. Dificultad Baja.

43. [Clase `SecuenciaCaracteres`: Elimina ocurrencias eficiente] Se pide modificar la solución del ejercicio 42 [Clase `SecuenciaCaracteres`: Elimina ocurrencias ineficiente] para que elimine las ocurrencias de un carácter de forma eficiente, tal y como se hizo en el ejercicio 17 [Elimina ocurrencias de una componente -versión eficiente-] de la Relación de Problemas III. Puede encontrar la solución a dicho ejercicio (trabajando sobre un vector declarado en el programa principal) en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/III_EliminaOcurrenciasEficiente.cpp

El método debe modificar la secuencia sobre la que se aplica.

Aplice este método para eliminar todos los espacios en blanco en el texto del Quijote, redirigiendo la entrada de datos (tal y como se indica en la página 40):

<https://decsai.ugr.es/jccubero/FP/Quijote.txt>

Redirija también la salida de datos a un fichero, para que no se muestre en pantalla el resultado (lo cual podría ralentizar la ejecución del programa)

Puede usar el mismo esbozo del programa del ejercicio 42 [Clase *SecuenciaCaracteres: Elimina ocurrencias ineficiente*] :

https://decsai.ugr.es/jccubero/FP/IV_SecCaract_EliminaOcurrenciasIneficienteEsbozo.cpp

Finalidad: Trabajar con un vector dentro de una clase. Dificultad Media.

44. [Clase *SecuenciaCaracteres: Vocal*] Recupere la clase *SecuenciaCaracteres* disponible en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/IV_SecuenciaCaracteres.cpp

Queremos ver si alguna de las letras de la secuencia es una vocal. Se plantean las siguientes cabeceras:

- `bool EsVocal(int indice)`
- `bool EsVocal(char caracter)`

Justifique, para cada una de las dos cabeceras anteriores, si tendría sentido que fuesen una función global, un método público o un método privado e indique cómo sería la llamada desde el programa principal (o desde cualquier sitio en general). No hace falta que cree un programa principal de prueba ni tampoco hace falta que implemente el método.

Finalidad: Diseño de la interfaz pública de una clase. Dificultad Baja.

45. [Túnel] El enunciado de este ejercicio es largo. Léalo por completo antes de empezar a plantear la solución. Al final del ejercicio podrá encontrar un enlace para descargar una parte ya implementada del programa que se le pide.

Se quiere construir la clase `Tune1` para gestionar la información de los vehículos que entran y salen por un túnel en un día. La información que se va a leer desde la entrada por defecto es la siguiente:

- En primer lugar aparece la longitud en Km del túnel (como un dato de tipo `double`)
- A continuación, los movimientos de los coches de la siguiente forma:
 - Se leerá un `char` indicando el tipo de movimiento: 'E' si es una entrada y 'S' si es una salida. El carácter '#' indicará la finalización de los datos de entrada al programa. En el caso de que el carácter no sea ni 'E' ni 'S', se considera que es un error en la entrada de datos y finalizará la ejecución del programa.
 - A continuación se leerá un string con la matrícula del vehículo
 - Finalmente, se leerá el instante en el que se produce el movimiento (la entrada o la salida). Vendrá dado en el formato horas, minutos, segundos (tres datos de tipo `int`). Se supone que los datos de entrada son correctos y los enteros están en los rangos correspondientes (las horas entre 0 y 23 y los minutos y segundos entre 0 y 59)

Por ejemplo:

```
3.4          -> Longitud en Km del túnel
E 4733MTI 0 0 13 -> El vehíc. 4733MTI entra en el inst. 0h0m13s
E 5232LTL 0 1 19 -> El vehíc. 5232LTL entra en el inst. 0h1m19s
S 4733MTI 0 1 36 -> El vehíc. 4733MTI sale en el inst. 0h1m36s
E 3330PRB 0 2 40 -> El vehíc. 3330PRB entra en el inst. 0h3m14s
#             -> Fin entrada datos
```

Se pide construir la clase `Tune1` acorde a lo siguiente:

- Debe tener en cuenta la longitud en km del túnel. Debe elegir si se especifica en el constructor o en algún método.
- Las entradas y salidas de los vehículos al túnel, se indicarán ejecutando sendos métodos `Entra` y `Sale` de la clase `Tune1`. Ambos métodos recibirán como parámetros la matrícula del vehículo y el instante (dado como tres enteros representando horas, minutos, segundos)

Por ejemplo, si tenemos un objeto `tune1` y el vehículo con matrícula "4733MTI" entra en el instante 0 0 13, habrá que ejecutar la sentencia

```
tune1.Entra("4733MTI", 0, 0, 13);
```

Si ese mismo vehículo sale en el instante 0 1 36, habría que ejecutar la sentencia

```
tunel.Sale("4733MTI", 0, 1, 36);
```

- Declare dentro de la clase `Tunel` un dato miembro que será un vector de `string` `matriculas` que almacenará las matrículas de todos los coches que vayan entrando. También debe definir dos vectores `entradas`, `salidas` que van a almacenar la información temporal de las entradas y salidas de cada vehículo (también serán datos miembro). En vez de almacenar tres enteros por cada entrada y por cada salida, se almacenarán los segundos totales transcurridos. Por ejemplo, si un vehículo entra en el instante 0 1 19, se almacenará 79 en la correspondientes componente del vector `entradas`. Si un vehículo sale en el instante 0 1 36, se almacenará 96 en la correspondientes componente del vector `salidas`.

Al principio, todas las componentes de los vectores se pondrán con un valor `NULO` (elija como valor `NULO` aquel que le parezca más adecuado) Cada vez que entre un vehículo, se añadirá la matrícula al vector `matriculas` y los segundos totales del instante correspondiente al vector `entradas`. Cuando salga un vehículo, se modificará el valor nulo que había en la correspondiente componente del vector `salidas` y se le asignará a esa componente los segundos totales del instante de salida. Observe que cuando un vehículo sale del túnel, no se elimina ninguna componente de ningún vector sino que únicamente se modifica la componente del vector `salidas`.

Siguiendo el anterior ejemplo, después de haber introducido los datos, el objeto de la clase `Tunel` contendrá lo siguiente:

```
num_vehiculos_tunel: 3

matriculas -> 4733MTI    5232LTL    3330PRB
entradas    -> 13        79          160
salidas      -> 96        NULO        NULO
```

Para calcular el número de segundos totales transcurridos a partir de los tres enteros del instante (horas, minutos, segundos), utilice la clase `Instante` vista en el ejercicio 36 [Instante (implementación)] . Puede usar la implementación de la clase `Instante` que se encuentra en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/IV_Instante.cpp

- Añada los métodos que estime oportuno a la clase `Tunel` para poder consultar los datos almacenados de los vehículos que hay en el túnel.

Una vez leídos todos los datos, el programa mostrará un informe mostrando las matrículas de cada vehículo y su velocidad media en el túnel (en el caso de que hayan salido). Para ello, defina los métodos que estime oportuno.

Con los datos del ejemplo anterior, la salida correcta sería:

RELACIÓN DE PROBLEMAS DEL TEMA IV. Funciones y Clases

Matrícula: 4733MTI
Velocidad: 147.5 km/h

Matrícula: 5232LTL
Velocidad: No ha salido

Matrícula: 3330PRB
Velocidad: No ha salido

Para mostrar los datos de la velocidad (que es un double) use la clase `FormateadorDoubles` vista en el ejercicio 33 [Diseño de la interfaz de la clase `FormateadorDoubles`] y redondee a 1 cifra decimal. Puede usar la implementación de la clase `FormateadorDoubles` que se encuentra en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/IV_FormateadorDoubles.cpp

Si lo desea, puede utilizar el siguiente fichero que contiene un esbozo del programa principal:

https://decsai.ugr.es/jccubero/FP/IV_TunelEsbozo.cpp

Ejemplo de entrada:

```
3.4
E 4733MTI 0 0 13
E 5232LTL 0 1 19
S 4733MTI 0 1 36
E 3330PRB 0 2 40
S 5232LTL 0 3 25
#
```

— Salida correcta:

Matrícula: 4733MTI
Velocidad: 147.5 km/h

Matrícula: 5232LTL
Velocidad: 97.1 km/h

Matrícula: 3330PRB
Velocidad: No ha salido

En el siguiente enlaces puede encontrar un fichero con más datos de entrada:

https://decsai.ugr.es/jccubero/FP/datos_tunel.txt

con los resultados correctos incluidos en este enlace:

https://decsai.ugr.es/jccubero/FP/resultados_tunel.txt

RELACIÓN DE PROBLEMAS DEL TEMA IV. Funciones y Clases

Finalidad: Trabajar con vectores dentro de una clase. Construir un programa con varias clases. Dificultad Media.

46. [Distancias entre ciudades] (*Examen Febrero 2017*) Recupere la solución de los ejercicios 24 [Distancias entre ciudades] y 25 [Distancias entre ciudades (mejor escala)] de la Relación de Problemas III. Si lo desea, puede usar la solución propuesta en el siguiente enlace (que trabaja con una matriz dentro del programa principal):

https://decsai.ugr.es/jccubero/FP/III_Ciudades.cpp

Se desea construir la clase `MapaDistancias` para representar y gestionar la información anterior. La clase contendrá un dato miembro privado de tipo matriz de reales para almacenar dichas distancias. Debe construir los métodos que resuelven las tareas de los ejercicios 24 [Distancias entre ciudades] y 25 [Distancias entre ciudades (mejor escala)], a saber:

```
int CiudadMejorConectada()
int MejorEscalaEntre(int origen, int destino)
```

Muy importante: Puede (y debe) definir los métodos auxiliares que estime oportuno para resolver las tareas anteriores.

Puede usar el esbozo del programa que se encuentra en el siguiente enlace. El programa lee los datos de la matriz y a continuación los índices de dos ciudades origen y destino:

https://decsai.ugr.es/jccubero/FP/IV_CiudadesEsbozo.cpp

Ejemplo de entrada: 5 50 100 0 150 70 0 0 60 80 90 1 4

— Salida correcta:

```
La ciudad con más conexiones directas es la ciudad 2
La mejor escala entre 1 y 4 es la ciudad 2
```

Finalidad: Trabajar con una matriz dentro de una clase.. Dificultad Media.

Ejercicios complementarios sobre funciones

47. [Se dividen] Recupere la solución del ejercicio 2 [Se dividen] de la Relación de Problemas II cuyo código puede encontrar en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/II_SeDividen.cpp

Modifique dicho código para que el programa principal llame a una función `SeDividen` que compruebe si un valor entero divide a otro (o al revés)

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

48. [Mayor subsecuencia de temperaturas] Recupere la solución del ejercicio 33 [Secuencia de temperaturas] de la Relación de Problemas II. Puede encontrarlo en el siguiente enlace:

https://decsai.ugr.es/jccubero/FP/II_MayorSecuenciaTemperaturas.cpp

Defina una función que compruebe si una temperatura está en el rango correcto, es decir, dentro del intervalo $[MIN=-60, MAX=90]$. Modifique el código del programa principal para que realice las llamadas oportunas a esta función y así evitar duplicar el código que hacía las comparaciones con los extremos `MIN`, `MAX` del intervalo.

Finalidad: Diseño de una función. Dificultad Baja.

49. [DoubleToString mejorado] En el ejercicio 17 [DoubleToString] hemos usado la función `to_string` para convertir un `string` a un `double`. Un inconveniente de esta función es que siempre redondea a 6 cifras decimales. Por ejemplo, el resultado de `to_string(0.12345678)` es la cadena "0.123457".

Modifique la función `DoubleToString` del ejercicio 17 [DoubleToString] para que redondee un real a un número de cifras arbitrario y obtenga la cadena de caracteres correspondiente.

Aplice la siguiente idea. Iremos obteniendo los dígitos y con la función `to_string` aplicada sobre dichos dígitos iremos construyendo la cadena de caracteres. Supongamos el real $r = 12.345678$ y un número de cifras decimales $n = 3$. Los pasos a seguir serían los siguientes:

- Trunque el real y obtenga el entero 12. Aplique la función `to_string` para obtener "12" y concatene con "." para obtener la cadena "12."
- Haga lo siguiente con la parte real de r
 - Redondee a n cifras decimales $\rightarrow 0.346$
 - Obtenga la primera cifra del real anterior $\rightarrow 3$

RELACIÓN DE PROBLEMAS DEL TEMA IV. Funciones y Clases

- Aplique al valor anterior la función `to_string` --> "3" y concatene el resultado en la cadena --> "12.3"
- Vuelva a hacer lo mismo con el real obtenido al desplazar la coma una posición a la derecha --> `r = 3.46`

Escriba un programa de prueba como el indicado en el ejercicio 17 [DoubleToString]

Ejemplo de entrada: 3.49 1 — Salida correcta: "3.5"

Ejemplo de entrada: 3.49 2 — Salida correcta: "3.49"

Ejemplo de entrada: 3.496 2 — Salida correcta: "3.5"

Ejemplo de entrada: 0.12345678 7 — Salida correcta: "0.1234568"

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

50. [Sistema de Hondt] Recupere la solución del ejercicio 12 [Sistema de D'Hondt] . Re-escribalo utilizando una función para calcular el cociente de Hondt.

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

51. [Robot] (Examen Enero 2020) Se dispone de un robot que se mueve en una línea, utilizando pasos de longitud fija. El robot se ubica en una posición `pos`, siendo `pos` un valor entero $1 \leq pos \leq 100$. Luego, el robot ejecuta una serie de órdenes, indicadas mediante un vector `ordenes` de tipo `char`. Cada orden es una letra 'I' o 'D', indicando si el robot se mueve a la izquierda (decrementando la posición actual `pos` en 1 unidad) o a la derecha (incrementando la posición actual `pos` en 1 unidad). Si es una letra distinta, el robot permanecerá en la misma posición.

Las posiciones válidas del robot cumplen $1 \leq pos \leq 100$. Se dice que una serie de órdenes es correcta si el robot nunca se sale de las posiciones válidas.

Se pide implementar un programa para que, dada una posición inicial y un vector de órdenes haga lo siguiente:

- Si la serie de órdenes es correcta, muestre cuántas veces se visitó cada posición.
- Si la serie de órdenes NO es correcta, el programa terminará indicando cuántas órdenes se pudieron ejecutar.

Escriba un programa que lea la posición inicial y a continuación los caracteres de las órdenes y los almacene en un vector de caracteres. El final de datos se indicará introduciendo el literal '+ '.

- En el caso de que la secuencia de órdenes sea correcta, el programa imprimirá el número de veces que se ha visitado cada posición.
- En otro caso, el programa imprimirá el número de órdenes que pudieron ejecutarse

Para resolver este problema, debe definir las funciones que estime oportuno para no duplicar código.

Ejemplo de entrada: 250 DDIIII+

— Salida correcta: Posición inicial incorrecta

Ejemplo de entrada: 10 DDIIII+

— Salida correcta: Frecuencia de visitas por posición: (8,1), (9,1), (10,2), (11,2), (12,1)

Ejemplo de entrada: 1 DIID+

— Salida correcta: Serie de órdenes incorrecta. Se ejecutaron 2 órdenes

Finalidad: Definición de funciones sencillas para no duplicar código. Dificultad Baja.

52. [Es letra del alfabeto español] Defina una función que compruebe si un dato de tipo `char` es una letra del alfabeto español. Debe considerar la 'ñ' así como las vocales mayúsculas y minúsculas acentuadas.

Escriba un sencillo programa de prueba.

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

53. Reescriba la solución del ejercicio 76 que calcula la suma de los primeros T factoriales. Para ello, debe leer el valor T usando la función `LeeIntRango` del ejercicio 13 para obligar a que esté en el intervalo $[1, 20]$.

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

- En una primera versión, la función `SumaFactoriales` debe llamar a la función `Factorial`, para realizar la suma tal y como se indica en el ejercicio 76
- En una segunda versión, la función `SumaFactoriales` debe realizar la suma de forma directa tal y como se indica en el ejercicio 77. Ponga dentro de un comentario la primera versión.

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

54. Retome la solución del ejercicio 63 (Gaussiana) y modifíquela introduciendo funciones dónde crea conveniente. Al menos debe definir la función `gaussiana` para que calcule el valor de la ordenada, para unos valores concretos de abscisa, esperanza y desviación.

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

55. Retome la solución del ejercicio 54 (Gaussiana)

Ahora estamos interesados en obtener el área que cubre la función gaussiana en el intervalo $[-\infty, x]$. Dicho valor se conoce como la *distribución acumulada (cumulative*

distribution function) en el punto x , abreviado $CDF(x)$. Matemáticamente se calcula realizando la integral:

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

Puede probar algunos valores ejecutando la siguiente calculadora online:

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

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

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

$$CDF(x) = \text{Área hasta } (x) \approx 1 - \text{gaussiana}_{0,1}(x)(b_1t + b_2t^2 + b_3t^3 + b_4t^4 + b_5t^5)$$

dónde:

$$t = \frac{1}{1 + b_0x} \quad b_0 = 0.2316419 \quad b_1 = 0.319381530 \quad b_2 = -0.356563782$$

$$b_3 = 1.781477937 \quad b_4 = -1.821255978 \quad b_5 = 1.330274429$$

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

Modifique el programa principal del ejercicio 54 para que llame a la función $CDF(x)$ e imprima las ordenadas correspondientes a las abscisas *minimo*, *minimo + incremento*, *minimo + 2*incremento*, etc.

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

-- Salida correcta:

f(11)=0.0782085

CDF(11)=0.998414

f(11.5)=0.0793905

CDF(11.5)=0.998573

f(12)=0.0797885

CDF(12)=0.998727

f(12.5)=0.0793905

CDF(12.5)=0.998875

f(13)=0.0782085

CDF(13)=0.999016

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

56. Retome la solución del ejercicio 52 (parking) de la Relación de Problemas II. Se quiere extender para poder trabajar con varios parkings o con varias tarifas distintas. Supondremos que en todos los casos, se tiene el mismo número de tramos (4) aunque puede variar la cuantía a tarifar por minutos y los límites de cada uno de los tramos.

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

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

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

Por ejemplo:

```
30      -> Limite 1 del parking 1
90      -> Limite 2 del parking 1
120     -> Limite 3 del parking 1
660     -> Limite 4 del parking 1
0.0412  -> Tarifa Tramo 1 del parking 1
0.0370  -> Tarifa Tramo 2 del parking 1
0.0311  -> Tarifa Tramo 3 del parking 1
0.0305  -> Tarifa Tramo 4 del parking 1
31.55   -> Tarifa día completo del parking 1
35      -> Limite 1 del parking 2
85      -> Limite 2 del parking 2
110     -> Limite 3 del parking 2
660     -> Limite 4 del parking 2
0.0402  -> Tarifa Tramo 1 del parking 2
0.0375  -> Tarifa Tramo 2 del parking 2
0.0319  -> Tarifa Tramo 3 del parking 2
0.0315  -> Tarifa Tramo 4 del parking 2
32      -> Tarifa día completo del parking 2
2 1 30  -> Entra a las 2 de la madrugada, 1 minuto, 30 segundos
4 2 50  -> Sale a las 4 de la madrugada, 2 minutos y 50 segundos
2 2 5   -> Entra a las 2 de la madrugada, 2 minutos, 5 segundos
4 3 7   -> Sale a las 4 de la madrugada, 3 minutos, 7 segundos
-1      -> Fin de la entrada de datos.
```

RELACIÓN DE PROBLEMAS DEL TEMA IV. Funciones y Clases

Ejemplo de entrada:

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

— Salida correcta:

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

```
47.1755
47.731
```

Finalidad: Diseño de una función. Dificultad Media.

57. Retome la solución del ejercicio 66 (población) de la Relación de Problemas II. Re-escribalo usando las funciones `LeeIntRango` del ejercicio 13 para leer los valores de las tasas y `LeeIntMayorIgualQue` del ejercicio 14 para leer el número de años que sea positivo. Defina también sendas funciones para calcular los dos valores que se piden en el ejercicio, a saber, el número de habitantes después de tres años y el número de años que pasarán hasta doblar la población inicial. Intente diseñar las funciones para que sean lo más generales posible.

Ejemplo de entrada:

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

— Salida correcta: 1490027497 27 2824131580

Finalidad: Diseño de una función. Dificultad Baja.

58. Retome la solución de los ejercicios 53 y 68 (servicio atención telefónica) de la Relación de Problemas II. Recordemos que el criterio de subida salarial era el siguiente:

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

Grado de satisfacción ≥ 4 :	+2%
----------------------------------	-----

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

Al igual que se pedía en el ejercicio 68 debe ir leyendo los datos de tres empleados en el siguiente orden:

RELACIÓN DE PROBLEMAS DEL TEMA IV. Funciones y Clases

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

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

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

Puede utilizar el fichero de datos `datos_atencion_telefonica.txt` disponible en [PRADO](#) . La salida correcta para este fichero es 1016.196 118.287 128.893

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

59. Retome la solución del ejercicio 58 de esta Relación de Problemas. Modifíquela para tener en cuenta que los límites correspondientes a los casos resueltos (20 y 30) y el grado de satisfacción media (4), así como los porcentajes de incrementos correspondientes (3%, 4% y 2%) ya no son constantes sino que pueden variar.

Por lo tanto, debe leer desde teclado dichos valores límites (justo después del salario por hora y en el orden indicado anteriormente) y cambiar la función definida en el ejercicio 58 para que tenga en cuenta este cambio.

Puede utilizar el fichero de datos

`datos_atencion_telefonica_limites_variables.txt`

disponible en [PRADO](#) . La salida correcta para este fichero es 1016.196 118.287 128.893

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

60. Implemente la solución del ejercicio 27 (Narcisista) de la relación de problemas II, usando funciones.

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

61. Escriba una función en C++ `LeeOpcion2Alternativas` que imprima en pantalla un mensaje, lea una opción como un carácter y sólo permita aceptar los caracteres 'S' o 'N' (mayúscula o minúscula). ¿Qué debería devolver la función? ¿El carácter leído o un `bool`? Aplique esta función en la solución del ejercicio 57 (Renta bruta y

neta) de la relación de problemas II, para leer si una persona es pensionista o si es autónomo.

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

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

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

63. Cree las siguientes funciones relacionadas con la progresión geométrica que se vio en el ejercicio 80 de la relación de problemas II. Analice cuáles deben ser los parámetros a estas funciones.

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

Para implementarla, use el mismo algoritmo (con un bucle `for`) que se vio como solución del ejercicio 80 de la relación de problemas II.

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

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

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

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

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

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

Cree un programa principal que llame a estas funciones.

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

64. Amplie el ejercicio 63 cambiando la implementación de la función `SumaHasta`. Para ello, en vez de usar un bucle aplicamos la siguiente fórmula que nos da la sumatoria aplicando únicamente cinco operaciones:

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

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

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

Hay que destacar que el cómputo de la potencia es una operación costosa, por lo que hasta podría ser más lenta la versión nueva que la antigua usando un bucle `for`. Probad distintos valores para ver si hay diferencias significativas. En cualquier caso, lo importante es que mientras no cambiemos la cabecera de la función `SumaHasta`, podemos cambiar su implementación sin tener que cambiar ni una línea de código del `main`.

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

65. Se pide construir las siguientes funciones:

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

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

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

La cabecera de la función será la siguiente:

```
char LeeMayuscula()
```

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

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

Puede suponer que no se introducen espacios en blanco.

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

RELACIÓN DE PROBLEMAS DEL TEMA IV. Funciones y Clases

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

66. Recupere la solución del ejercicio 18 de la Relación de Problemas II (pasar de mayúscula a minúscula y viceversa usando un enumerado) Para que el tipo de dato enumerado sea accesible desde dentro de las funciones, debemos ponerlo antes de definir éstas, es decir, en un ámbito global a todo el fichero. Se pide definir las siguientes funciones y cread un programa principal de ejemplo que las llame:

- a) `Capitalizacion` nos dice si un carácter pasado como parámetro es una minúscula, mayúscula u otro carácter. A dicho parámetro, llamadlo `una_letra`. La función devuelve un dato de tipo enumerado.
- b) `Convierte_a_Mayuscula` comprueba si un carácter pasado como parámetro es minúscula (para ello, debe llamar a la función `Capitalizacion`), en cuyo caso lo transforma a mayúscula. Si el carácter no es minúscula debe dejar la letra igual. A dicho parámetro, llamadlo `caracter`.

Esta función hace lo mismo que la función `tolower` de la biblioteca `cctype`

Observad que el parámetro `una_letra` de la función `Capitalizacion` podría llamarse igual que el parámetro `caracter` de la función `Convierte_a_Mayuscula`. Esto es porque están en ámbitos distintos y para el compilador son dos variables distintas. Haced el cambio y comprobarlo.

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

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

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

- d) `CambiaMayusculaMinuscula`, a la que se le pase como parámetro un `char` y haga lo siguiente:
 - si el argumento es una letra en mayúscula, devuelve su correspondiente letra en minúscula,
 - si el argumento es una letra en minúscula, devuelve su correspondiente letra en mayúscula,
 - si el argumento no es ni una letra mayúscula, ni una letra mayúscula, devuelve el carácter pasado como argumento.

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

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

RELACIÓN DE PROBLEMAS DEL TEMA IV. Funciones y Clases

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

Realice un programa que implemente estas dos tareas:

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

Utilice las funciones que estime oportuno.

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

68. Defina una función para implementar la solución del ejercicio 82 de la relación de problemas II (Serie)

Dificultad Media.

69. Defina una función para implementar la solución del ejercicio 62 de la relación de problemas II (número feliz)

Dificultad Media.

Ejercicios complementarios sobre clases

70. [Coordenadas geográficas] Se quiere construir una clase `CoordenadasGPS` para representar las coordenadas geográficas de una posición terrestre, dada por tres datos reales, a saber, su longitud, latitud y altura, tal y como se describe en el ejercicio 44 [Coordenadas geográficas (distancia)] de la Relación de Problemas II. La clase debe proporcionar métodos para asignar y recuperar los datos de latitud y longitud. Se pide hacerlo de la siguiente forma:

- Sólo se podrán asignar los datos de longitud y latitud en grados. El/Los método/s correspondiente/s transformarán esos grados en radianes y los asignarán a los datos miembros.

La altura vendrá en metros.

Debe comprobar que los grados sean correctos, es decir, los grados de latitud deben estar en el intervalo $[-90, 90]$ y los de longitud en $[-180, 180]$. La altura debe estar entre -423 (Valle del Jordán) y 8848 (monte Everest).

- Para mostrar los datos de longitud y latitud se definirán métodos específicos para cada caso:

`LatitudGrados, LatitudRadianes,`
`LongitudGrados, LongitudRadianes`

Por ahora, la clase sólo debe proporcionar los métodos para asignar y recuperar los valores de latitud, longitud y altura. Posteriormente se ampliará la funcionalidad de esta clase.

Finalidad: Diseño de una clase básica. Dificultad Baja.

71. [Busca minas] (*Examen Septiembre 2015*) Éste es un juego muy conocido cuyo objetivo es encontrar todas las minas existentes en un tablero rectangular, sin abrir ninguna. En este ejercicio se va a crear una clase para almacenar el tablero (pero no se va a implementar nada relacionado con el desarrollo de una partida)

Se pide definir la clase `TableroBuscaMinas` conteniendo lo siguiente:

- a) Para representar el tablero se trabajará con una matriz de datos de tamaño máximo 50×30 en la que todas las filas tienen el mismo número de columnas y los datos son de tipo `bool`. Contendrá un valor `true` en caso de haber una mina en la casilla especificada y `false` en caso contrario. Esta matriz será un dato miembro de la clase y al principio, todos los valores estarán a `false`.
- b) Defina un método para incluir una mina en una determinada casilla.

- c) Defina un método que reciba las coordenadas (i, j) de una casilla y devuelva un valor entero que indique el número de minas que rodean a la misma (será un número entre 0 y 8). En caso de que la casilla contenga una mina, se devolverá el valor -1 .

En la implementación de este método ha de tener en cuenta que las casillas que hay en los *bordes* de la matriz no tienen 8 vecinos. Por ejemplo, la casilla en la posición $[0][0]$ sólo tiene tres vecinos rodeándola.

Construya un programa que lea un entero representando el número n de minas a generar (tendrá que ser un valor entre 1 y el número total de elementos de la matriz). A continuación, el programa asignará aleatoriamente n minas (utilice la clase generadora de números enteros pseudoaleatorios descrita en el ejercicio 30 [Generador aleatorio] de esta Relación de Problemas)

Una vez generadas las minas, recorra el tablero completo e imprima, por cada casilla, cuántas tiene a su alrededor. Por ejemplo, si el tablero contuviese los siguientes datos (0 representa false y 1 representa true)

Tablero original:

1	0	0
1	1	0
0	1	0
0	1	1
0	0	1

entonces, el resultado sería el siguiente:

Minas alrededor de cada casilla:

-1	3	1
-1	-1	2
4	-1	4
2	-1	-1
1	3	-1

Finalidad: Trabajar con matrices dentro de clases. Dificultad Baja.

72. [Tarifa aérea con clases] Recupere la solución del ejercicio 41 [Tarifa aérea: múltiples billetes] disponible en:

https://decsai.ugr.es/jccubero/FP/II_TarifaAereaMultiplesBilletes.cpp

Re-escribalo usando clases. Para ello, se pide que defina la clase `TarifaAerea` teniendo en cuenta que la tarifa base es de 150 euros por defecto pero podría especificarse otra cantidad distinta.

RELACIÓN DE PROBLEMAS DEL TEMA IV. Funciones y Clases

En el diseño de la clase deberá decidir en qué sitio (en el constructor o en los métodos) se establecen los valores de la tarifa base, la distancia del trayecto, el número de puntos del cliente, etc.

Finalidad: Diseño de una clase. Dificultad Media.

73. [Calificación final] Defina la clase `CalificacionFinal` para calcular la nota en la convocatoria ordinaria de Febrero de un alumno en la asignatura de Fundamentos de Programación. Para ello, debe considerar lo siguiente:

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

Estos porcentajes son los considerados por defecto. En cualquier caso, se quiere contemplar la posibilidad de manejar otros distintos.

- El profesor de cada grupo, tiene la posibilidad, si así lo desea, de subir la nota del examen escrito. Dicha subida no puede ser mayor de 0.5 puntos y puede variar entre grupos distintos, aunque, obviamente, será la misma para todos los alumnos de un mismo grupo. En cualquier caso, la subida sobre el examen escrito sólo se aplica a aquellos alumnos que, después de realizar la subida, saquen un 5 o más.

- Para poder aprobar la asignatura, es preciso haber sacado al menos un 4 en la nota del examen escrito (esta restricción se aplica antes de la subida de nota especificada en el apartado anterior).

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

El 4 es el límite por defecto. Al igual que los porcentajes del primer apartado, se quiere contemplar la posibilidad de manejar otros distintos.

Construya un programa principal que lea los datos en el siguiente orden (no hay límite en el número de grupos que se van a introducir):

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

RELACIÓN DE PROBLEMAS DEL TEMA IV. Funciones y Clases

El programa debe imprimir la nota final de cada alumno utilizando dos posibilidades:

- a) Suponiendo que las ponderaciones son 10 %, 10 %, 20 % y 60 % y la nota mínima para aprobar un 4
- b) Suponiendo que las ponderaciones son 5 %, 5 %, 20 % y 70 % y la nota mínima para aprobar un 4.4

En la dirección

https://decsai.ugr.es/jccubero/FP/notas_parciales.txt

se encuentra un fichero de prueba para este ejercicio. Las notas finales que el programa debería obtener se encuentran en:

https://decsai.ugr.es/jccubero/FP/notas_finales.txt

Ejemplo de entrada:

```
0.5
5.7  4.9  5.2  5.5
6.3  7.1  5.1  8.1
9.5  9.8  8    9.3
.....
-1
0.4
4.2  4.7  4.2  4.9
4    4.5  4.8  4.3
6.8  7.3  5.5  5.7
.....
-1
-1
```

— Salida correcta:

```
9.41/9.425
1.4/1.4
6.28/6.27
.....
4.91/4.995
4.39/4.3
6.17/6.075
.....
```

Finalidad: Diseño de una clase. Dificultad Media.

74. [Tabla de temperaturas] Se desea construir una clase `TablaTemperaturas` para almacenar y gestionar las temperaturas de 10 ciudades tomadas cada hora durante un

RELACIÓN DE PROBLEMAS DEL TEMA IV. Funciones y Clases

día. Por lo tanto, internamente trabajaremos con una matriz de 10 filas y 24 columnas de datos de tipo `double`. Defina, al menos, los siguientes métodos:

- `Valor` para obtener la temperatura de una ciudad (dada por un índice entero entre 0 y 9) a una hora determinada (será otro entero entre 0 y 23)
- `Modifica` para modificar la temperatura de una ciudad a una hora determinada.
- `Minimo` para obtener la mínima temperatura de una ciudad dada. Como siempre ocurre cuando trabajamos con vectores y matrices, el método debe devolver la columna (hora) en la que se alcanza, en vez del valor de la temperatura.
- `MaxMinimos` para obtener la ciudad y la hora en la que se alcanzó la máxima temperatura de entre los mínimos alcanzados por cada ciudad. El método debe devolver un `struct` del tipo

```
struct ParFilaColumna{
    int fila;
    int columna;
}
```

con la fila y columna en la que se alcanza el máximo de los mínimos. El método debe calcular el mínimo de cada fila (ciudad) y luego el máximo de todos ellos (de forma análoga a lo pedido en el ejercicio 28 [Máximo de los mínimos] de la Relación de Problemas III)

En el ejercicio 28 [Máximo de los mínimos] de la Relación de Problemas III se vio una solución eficiente, pero ahora queremos implementar otra solución no tan optimizada pero que nos permita trabajar con vectores locales y reutilizar otros métodos. En concreto, se pide que el método `MaxMinimos` implemente el siguiente algoritmo:

```
Recorrer todas las ciudades
Llamar al método Minimo para calcular el mínimo
de cada ciudad.
Guardar los resultados en un vector de mínimos
Recorrer el vector de mínimos para calcular el máximo
```

Construya un programa que lea las 24 temperaturas de 10 ciudades e imprima la fila y columna en la que se alcanza el máximo de los mínimos. Puede comprobar que la solución es correcta usando como datos de entrada los del siguiente fichero:

https://decsai.ugr.es/jccubero/FP/datos_temp.txt

La salida debe ser:

```
Ciudad: 2
Hora: 7
Temperatura: 16.2
```

Finalidad: Devolución de un struct. Vectores locales. Llamadas entre métodos. Dificultad Baja.

75. [Elimina repetidos ineficiente] Sobre la clase `SecuenciaCaracteres`, añada un método `EliminaRepetidos` que quite los elementos repetidos, de forma que cada componente sólo aparezca una única vez. Se mantendrá la primera aparición, de izquierda a derecha. Por ejemplo, si la secuencia contiene
`{'b','a','a','h','a','a','a','a','c','a','a','a','g'}`
después de quitar los repetidos, ésta quedaría como sigue:
`{'b','a','h','c','g'}`

Observe que se pide explícitamente que el método modifique los datos contenidos en la secuencia.

Implemente los siguientes algoritmos para resolver este problema:

- a) Usando un **vector local** `sin_repetidos` dentro del método `EliminaRepetidos`. En este vector local iremos almacenando la primera ocurrencia de cada carácter. Una vez terminemos de añadir convenientemente caracteres al vector `sin_repetidos`, lo volcamos en `vector_privado`. Nos quedaría:

```
Recorrer todas las componentes de "vector_privado"
Si la componente NO está en "sin_repetidos",
la añadimos a "sin_repetidos"
Volcar "sin_repetidos" en "vector_privado"
```

- b) El problema del algoritmo anterior es que usa un vector local, lo que podría suponer una carga importante de memoria si trabajásemos con vectores grandes. Por lo tanto, vamos a resolver el problema sin usar vectores locales. Defina otra versión del método `EliminaRepetidos` que implemente el siguiente algoritmo:

```
Recorrer todas las componentes de "vector_privado"
Si NO es la primera aparición de la componente,
Eliminarla
```

Para comprobar que no sea la primera aparición, basta buscarla a su izquierda (en la parte del vector privado que hay a su izquierda). Para eliminarla, basta llamar al método `Elimina` (de un único carácter). La implementación del método `Elimina` se encuentra en

https://decsai.ugr.es/jccubero/FP/IV_SecuenciaCaracteres.cpp

Construya un programa que lea los caracteres de la cadena uno a uno con `cin.get()`, hasta que se introduzca el carácter `#` y muestre el resultado de quitarle los repetidos.

Ejemplo de entrada: `ggabaabghc#` — Salida correcta: `gabhc`

Finalidad: Uso de vectores locales dentro de los métodos. Reutilización de métodos. Dificultad Media.

76. [Elimina repetidos eficiente] Recupere la solución del ejercicio 75 [Elimina repetidos ineficiente]. El algoritmo visto en dicho ejercicio, nos obliga a desplazar muchas componentes cada vez que encontremos una repetida. Proponga una alternativa (sin usar vectores locales) para que el número de desplazamientos sea el menor posible e implementela.

Consejo: Use la misma técnica que se indicó en el ejercicio 17 [Elimina ocurrencias de una componente -versión eficiente-] de la Relación de Problemas III.

Puede probar la diferencia radical en el tiempo de ejecución de los algoritmos anteriores con el texto del Quijote, disponible en:

<https://decsai.ugr.es/jccubero/FP/Quijote.txt>

Mientras que la versión eficiente tarda algunos *milisegundos*, la versión ineficiente puede tardar *una hora* (en un i7)

Finalidad: Desarrollo de algoritmos eficientes. Dificultad Media.

77. [Elimina ocurrencias entre dos posiciones] Se pide modificar la solución del ejercicio 43 [Clase SecuenciaCaracteres: Elimina ocurrencias eficiente] para que elimine todas las ocurrencias de un carácter que hay entre dos índices de la secuencia.

Por ejemplo, el resultado de eliminar todas las ocurrencias del carácter a entre las posiciones 2 y 6 de la secuencia abacdeaaafa sería la secuencia abcdeafa.

¿Mantendría los dos métodos, el visto en este ejercicio y el visto en el ejercicio 43 [Clase SecuenciaCaracteres: Elimina ocurrencias eficiente] ?

Finalidad: Trabajar con un vector dentro de una clase. Reutilizar un método. Dificultad Baja.

78. [Elimina ultimos] Sobre la clase SecuenciaCaracteres, construya un método EliminaUltimos para eliminar todos los caracteres que haya al final de la secuencia y que sean iguales a un carácter determinado. En definitiva, se desea realizar lo mismo que se implementó con funciones en el ejercicio 16 [Uso de funciones sobre cadenas], salvo que:

- En vez de trabajar con funciones a las que se le pasa como parámetro un string, trabajamos con métodos que actúan sobre un objeto de la clase SecuenciaCaracteres.
- Se desea modificar la secuencia de caracteres sobre la que se aplica el método EliminaUltimos.

Finalidad: Trabajar con un vector dentro de una clase. Dificultad Baja.

79. **[Elimina exceso de blancos]** Sobre la clase `SecuenciaCaracteres`, añada un método `EliminaExcesoBlancos` para eliminar el exceso de caracteres en blanco, es decir, que sustituya todas las secuencias de espacios en blanco por un sólo espacio. Por ejemplo, si la secuencia original es (' ', 'a', 'h', ' ', ' ', ' ', 'c'), que contiene una secuencia de tres espacios consecutivos, la secuencia resultante debe ser (' ', 'a', 'h', ' ', ' ', 'c').

Nota: Debe hacerse lo más eficiente posible.

Construya un programa que lea los caracteres de la cadena uno a uno con `cin.get()`, hasta que se introduzca el carácter # y muestre el resultado de quitarle el exceso de blancos. Puede probar el programa con el siguiente fichero, que contiene el Quijote con más de un espacio en blanco entre palabras:

https://decsai.ugr.es/jccubero/FP/Quijote_con_exceso_de_blan cos.txt

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

80. **[Circunferencia con structs]** Recupere la definición del struct `CoordenadasPunto2D` del ejercicio 101 de esta relación de problemas y la solución al ejercicio 3 (Circunferencia) de la relación de problemas I.

Cree ahora una clase llamada `Circunferencia`. Para establecer el centro, se usará un dato miembro que ha de ser de tipo `CoordenadasPunto2D`.

Añada métodos para obtener la longitud de la circunferencia y el área del círculo interior.

Añada también un método para saber si la circunferencia contiene a un punto cualquiera. Recordemos que un punto (x_1, y_1) está dentro de una circunferencia con centro (x_0, y_0) y radio r si se verifica que:

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

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

Cree un programa principal que lea el centro y el radio de una circunferencia, las coordenadas de un punto y muestre en pantalla la longitud de la circunferencia, el área del círculo y nos diga si el punto está dentro o no de la circunferencia.

Ejemplo de entrada: 2.1 3.2 5.8 2.2 4.6

— Salida correcta: 36.4425 105.683 El punto está dentro

Ejemplo de entrada: 2.1 3.2 5.8 2.2 10.36

— Salida correcta: 36.4425 105.683 El punto no está dentro

Finalidad: Trabajar con clases y el tipo struct. Dificultad Baja.

81. **[Login]** (*Examen Febrero 2013*) Recupere la solución del ejercicio 34 **[Login]** de la Relación de Problemas III. Implemente la clase `Login` y defina el método `Codifica` que recibirá una cadena de caracteres (tipo `string`) formada por el nombre y apellidos (separados por uno o más espacios en blanco) y devuelva otra cadena con la sugerencia de login.

```
class Login{
private:
    int num_caracteres_a_coger;
public:
    Login (int numero_caracteres_a_coger)
        :num_caracteres_a_coger(numero_caracteres_a_coger)
    { }
    string Codifica(string nombre_completo){
        .....
    }
};
```

Los únicos métodos que necesita usar de la clase `string` son `size` y `push_back`. Construya un programa que lea los caracteres de la cadena uno a uno con `cin.get()`, hasta que el usuario introduzca el carácter `#` y muestre el resultado de la codificación en pantalla.

Finalidad: Recorrido de las componentes de un vector, controlando qué ha ocurrido anteriormente. Dificultad Media.

82. **[Cuenta mayúsculas en una clase]** Sobre el ejercicio ?? de la Relación de Problemas III, construya una clase específica `ContadorMayusculas` que implemente los métodos necesarios para llevar el contador de las mayúsculas. Lo que se pretende es que la clase proporcione los métodos siguientes:

```
void IncrementaConteo (char mayuscula)
int  CuantasHay (char mayuscula)
```

El primer método aumentará en uno el contador de la correspondiente mayúscula y el segundo indicará cuántas hay. Modifique el programa principal para que cree un objeto de esta clase y llame a sus métodos para realizar los conteos de las mayúsculas. Finalmente, hay que imprimir en pantalla cuántas veces aparece cada mayúscula.

Finalidad: Diseño de una clase contadora de frecuencias. Dificultad Media.

83. Recupere la solución del ejercicio 55 de esta relación de problemas sobre la función gaussiana. En vez de trabajar con funciones, plantee la solución con una clase. Debe diseñar la clase teniendo en cuenta que la función matemática gaussiana viene determinada unívocamente por el valor de la esperanza y la desviación, es decir, son estos dos parámetros lo que distinguen a una función gaussiana de otra.

RELACIÓN DE PROBLEMAS DEL TEMA IV. Funciones y Clases

Finalidad: Diseño de una clase. Dificultad Baja.

84. Recupere la solución del ejercicio 57 de esta relación de problemas (población con funciones). Re-escribalo para que los cálculos relacionados con la población estén encapsulados en una clase. La lectura de los valores en los rangos adecuados se hará con las mismas funciones que ya se definieron en ese ejercicio. Modifique apropiadamente el programa principal.

Finalidad: Diseño de una clase. Dificultad Baja.

85. Recupere la solución del ejercicio 56 de esta relación de problemas (parking con funciones). Re-escribalo para que los cálculos relacionados con el cálculo de la tarifa, estén encapsulados en una clase. Mantenga la definición de la función `MinutosEntreInstantes` tal y como está. Modifique apropiadamente el programa principal.

Finalidad: Diseño de una clase. Dificultad Media.

86. En el ejercicio 63 de esta relación de problemas se definieron varias funciones para operar sobre una progresión geométrica. Defina ahora una clase para representar una progresión geométrica.

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

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

87. Recupere la solución del ejercicio 62 de esta relación de problemas (cálculo del salario en función de las horas trabajadas) Defina una clase `Nomina` para gestionar el cálculo del salario final. Suponga que el porcentaje de incremento en la cuantía de las horas extras (50 %) y el número de horas que no se tarifican como extra (40) son valores que podrían cambiar, aunque no de forma continua. El número de horas trabajadas y la cuantía a la que se paga cada hora extraordinaria, sí son cantidades que varían de un trabajador a otro.

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

88. Recupere la solución del ejercicio 57 (actualización de la retención fiscal) de la relación de problemas II. En este problema se leían caracteres de teclado ('s'/'n') para saber si una persona era autónomo, pensionista, etc.

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

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

89. Recuperela solución del ejercicio ?? (recta) de esta relación de problemas. Se pide crear un programa principal que haga lo siguiente:

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

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

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

90. Se quiere construir una clase `Nomina` para realizar la funcionalidad descrita en el ejercicio 34 de la relación de problemas I sobre la nómina del fabricante y diseñador (página Problemas-16). Cread los siguientes programas (entregad un fichero por cada uno de los apartados):

- a) Suponed que sólo gestionamos la nómina de una empresa en la que hay un fabricante y tres diseñadores. Los salarios brutos se obtienen al repartir los ingresos de la empresa, de forma que el diseñador cobra el doble de cada fabricante.
El programa leerá el valor de los ingresos totales y calculará los salarios brutos de los fabricantes y diseñador, llamando a los métodos oportunos de la clase `Nomina`.
- b) Supongamos que se aplica una retención fiscal y que ésta es la misma para los fabricantes y el diseñador. En el constructor se establecerá el porcentaje de retención fiscal (de tipo `double`) y posteriormente no se permitirá que cambie, de

forma que todas las operaciones que se hagan serán siempre usando la misma retención fiscal. Los salarios netos se obtienen al aplicar la retención fiscal a los salarios brutos (después de repartir los ingresos totales de la empresa):

```
salario_netos = salario_brutos -
                salario_brutos * retencion_fiscal / 100.0
```

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

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

- En cada sucursal hay un único diseñador pero el número de fabricantes es distinto en cada sucursal. Por tanto, el número de fabricantes habrá que especificarlo en el constructor y posteriormente no podrá cambiarse.
- La forma de repartir el dinero es la siguiente: el diseñador se lleva una parte del total y el resto se reparte a partes iguales entre los fabricantes. En los apartados anteriores, por ejemplo, la parte que se llevaba el diseñador era $2/5$ y el resto ($3/5$) se repartía entre los tres fabricantes. La parte que el diseñador se lleva puede ser distinta entre las distintas sucursales ($2/5$, $1/6$, etc), pero no cambia nunca dentro de una misma sucursal. Por tanto, el porcentaje de ganancia ($2/5$, $1/6$, etc) habrá que especificarlo en el constructor y posteriormente no podrá cambiarse.
- Las retenciones fiscales de los fabricantes y diseñador son distintas. Además, se prevé que éstas puedan ir cambiando durante la ejecución del programa. Por lo tanto, no se incluirán como parámetros en el constructor.

El programa leerá los siguientes datos desde un fichero externo:

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

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

```
2
300 3 6 20 18
400 5 4 22 19
```

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

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

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

La clase debe proporcionar métodos para cambiar la estrella y el piñón, sabiendo que la estrella avanza o retrocede de 1 en 1 y los piñones cambian a saltos de uno o de dos. Si ha llegado al límite superior (inferior) y se llama al método para subir (bajar) la estrella, la posición de ésta no variará. Lo mismo se aplica al piñón.

Cree un programa principal que lea desde un fichero externo los movimientos realizados e imprima la situación final de la estrella y piñón. Los datos se leerán en el siguiente formato: tipo de plato (piñón o estrella) seguido del tipo de movimiento. Para codificar esta información se usarán las siguientes letras: E indica una estrella, P un piñón, S para subir una posición, B para bajar una posición, T para subir dos posiciones y C para bajar dos posiciones. T y C sólo se aplicarán sobre los piñones.

E S P S P S P S P C E S E B #

En este ejemplo los movimientos serían: la estrella sube, el piñón sube en tres ocasiones sucesivas, el piñón baja dos posiciones de golpe, la estrella sube y vuelve a bajar. Supondremos siempre que la posición inicial de la estrella es 1 y la del piñón 1. Así pues, la posición final será Estrella=1 y Piñón=2.

Mejorad la clase para que no permita cambiar la marcha (con la estrella o el piñón) cuando haya riesgo de que se rompa la cadena. Este riesgo se produce cuando la marcha a la que queremos cambiar es de la siguiente forma:

- Estrella igual a 1 y piñón mayor o igual que 5
- Estrella igual a 2 y piñón o bien igual a 1 o bien igual a 7
- Estrella igual a 3 y piñón menor o igual que 3

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

92. Recuperad la solución del ejercicio 67 de la Relación de Problemas II (Empresa). Reescribid el programa principal usando una clase *Ventas* para gestionar los cálculos de las ventas realizadas. Únicamente se pide que se indiquen las cabeceras de los métodos públicos de la clase y las llamadas a éstos en el programa principal. No hay que implementar ninguno de los métodos.

Debe suponer que la clase gestionará las ventas de exactamente tres sucursales. Los códigos de dichas sucursales son enteros cualesquiera (no necesariamente 1, 2, 3, como ocurría en el ejercicio 67 de la Relación de Problemas II)

El programa principal sería de la siguiente forma:

```
Ventas ventas_empresa;
.....
while (identif_sucursal != TERMINADOR){
    cin >> cod_producto;
    cin >> unidades_vendidas;

    --> Actualiza el número de unidades
        vendidas de la sucursal leída
        llamando a un método de ventas_empresa

    cin >> identif_sucursal;
}

--> Obtener el identificador y el número de ventas
    de la sucursal ganadora llamando a un método
    de ventas_empresa
```

Finalidad: Diseño de una clase. Dificultad Media.

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

Finalidad: Diseño de una clase. Dificultad Media.

94. Implemente una clase para representar un número complejo. Un complejo se define como un par ordenado de números reales (a, b) , donde a representa la parte real y b la parte imaginaria. Construya un programa principal que lea la parte real e imaginaria, cree el objeto e imprima el complejo en la forma $a + bi$.

Por ahora no podemos implementar métodos para sumar, por ejemplo, dos complejos. Lo veremos en el último tema.

95. Una empresa quiere gestionar las nóminas de sus empleados. El cómputo de la nómina se realiza en base a los siguientes criterios:

- Hay cuatro tipos de categorías laborales: Operario, Base, Administrativo y Directivo.
- Se parte de un salario base que depende de la antigüedad del trabajador y de su categoría laboral. Para la categoría Operario, el salario base es de 900 euros, 1100 el puesto Base, 1200 los Administrativos y 2000 los Directivos. Dicho salario base se incrementa con un tanto por ciento igual al número de años trabajados.

- c) Los trabajadores tienen complementos en su nómina por el número de horas extraordinarias trabajadas. La hora se paga distinta según la categoría: 16 euros por hora para los operarios, 23 para el puesto Base, 25 los Administrativos y 30 los Directivos. Además, al complemento que sale al computar el número de horas extraordinarias, se le aplica una subida con un tanto por ciento igual al número de años trabajados.

Se pide diseñar la interfaz de una clase (también hay que incluir los datos miembro privados) para poder trabajar con esta información. No se pide implementar la clase, únicamente determinar la interfaz.

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

96. Implementad la clase del ejercicio 95 de esta relación de problemas. *Dificultad Media.*
97. Recuperad la solución del ejercicio 93 (Empresa) y modificadlo convenientemente para que los datos miembros que referencia los identificadores de las sucursales sean constantes.

Finalidad: Trabajar con datos miembros constantes. Dificultad Baja.

98. La sonda Mars Climate Orbiter fue lanzada por la NASA en 1998 y llegó a Marte el 23 de septiembre de 1999. Lamentablemente se estrelló contra el planeta ya que se acercó demasiado. El error principal fue que los equipos que desarrollaron los distintos módulos de la sonda usaron sistemas de medida distintos (el anglosajón y el métrico). Cuando un componente software mandaba unos datos en millas (o libras), otro componente software los interpretaba como si fuesen kilómetros (o Newtons). El problema se habría arreglado si todos hubiesen acordado usar el mismo sistema. En cualquier caso, cada equipo se encuentra más a gusto trabajando en su propio sistema de medida. Por tanto, la solución podría haber pasado por que todos utilizasen una misma clase para representar distancias (idem para fuerzas, presión, etc), utilizando los métodos que les resultasen más cómodos.

Para ello, se pide construir la clase `Distancia` que contendrá métodos como `SetKilometros`, `SetMillas`, etc. Internamente se usará un único dato miembro privado llamado `kilometros` al que se le asignará un valor a través de los métodos anteriores, realizando la conversión oportuna (una milla es 1.609344 kilómetros). La clase también proporcionará métodos como `GetKilometros` y `GetMillas` para lo que tendrá que realizar la conversión oportuna (un kilómetro es 0.621371192 millas).

Observad que la implementación de la clase podría haber utilizado como dato miembro privado, una variable `millas`, en vez de `kilómetros`. Esto se oculta a los usuarios de la clase, que sólo ven los métodos `SetKilometros`, `SetMillas`, `GetKilometros` y `GetMillas`.

Cread un programa principal que pida algunos datos y muestre los valores convertidos.

Nota. Otro de los fallos del proyecto fue que no se hicieron suficientes pruebas del software antes de su puesta en marcha, lo que podría haber detectado el error. Esto

pone de manifiesto la importancia de realizar una batería de pruebas para comprobar el correcto funcionamiento del software en todas las situaciones posibles. Esta parte en el desarrollo de un proyecto se le conoce como *pruebas de unidad (unit testing)*

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

99. Construid una clase llamada `MedidaAngulo` que represente una medida de un ángulo. Al igual que se hizo en el ejercicio 98, la clase aceptará datos que vengan de alguna de las siguientes formas: número de grados con decimales (real); número de radianes (entero); número de segundos (entero); número de grados, minutos y segundos (en un struct que represente estos tres valores)

Dificultad Baja.

100. Recupere la solución del ejercicio 85 (Parking con una clase) Defina un struct llamado `InstanteTiempo` para almacenar la hora, minutos y segundos que constituyen un instante de tiempo. Cambie la definición de la función `MinutosEntreInstantes` y el programa principal para que trabaje con este tipo struct.

Finalidad: Trabajar con funciones y el tipo struct. Dificultad Baja.

101. Defina un struct llamado `CoordenadasPunto2D` para representar un par de valores reales correspondientes a un punto en \mathbb{R}^2 .

Defina una función `DistanciaEuclidea` para que calcule la distancia entre dos puntos cualesquiera. Cree un programa principal que vaya leyendo 4 valores reales desde teclado representando las coordenadas de dos puntos y calcule la distancia euclídea entre ellos. Cada vez que se lean los cuatro valores se le preguntará al usuario si quiere seguir introduciendo datos o no (con las opciones 's'/'n').

Ejemplo de entrada:

```
s  3.1 4.2 5.3 6.4  j k s  2.1 4.9 -3.2 0    s  1 5 1 5  n
```

-- Salida correcta: 3.11127 7.21803 0

Finalidad: Trabajar con funciones y el tipo struct. Dificultad Baja.

102. [Parking] Recupere la solución del ejercicio 56 de la relación de problemas III (párking). Re-escribalo definiendo la clase `TarifadorParking` para calcular la tarifa.

La clase debe permitir cualquier número de tramos. Para ello, haga lo siguiente:

- Defina dos vectores como datos miembro de la clase. En uno almacenaremos los límites de los tramos y en el otro la correspondiente tarifa.
- Defina el siguiente método:

```
void AniadeTramo(double limite_superior_tramo,  
                double tarifa_tramo)
```

Este método se llamará tantas veces como tramos tengamos.

- Defina el método `GetTarifa` para calcular la tarifa según el número de minutos de un estacionamiento.
- Cree dos objetos de la clase `TarifadorParking` (uno para cada parking) y modifique adecuadamente el programa principal para calcular las tarifas a partir de los métodos de los objetos.

Mantenga la definición de la función `MinutosEntreInstantes` para calcular los minutos que hay entre dos instantes.

Finalidad: Diseñar las cabeceras de los métodos que acceden a las componentes del vector. Dificultad Baja.

103. **[Fibonacci]** La sucesión de Fibonacci de orden n es una secuencia de números en la que los dos primeros son el 0 y el 1. A partir del tercero, los elementos se calculan como la suma de los n anteriores, si ya hay n elementos disponibles, o la suma de todos los anteriores si hay menos de n elementos disponibles.

Por ejemplo, la sucesión de Fibonacci de orden 4 sería la siguiente:

0, 1, 1, 2, 4, 8, 15, 29, ...

Defina una clase llamada `Fibonacci`. Para almacenar los enteros, se usará un vector de enteros. Al constructor se le pasará como parámetro el valor de n . Defina los siguientes métodos:

- `int GetBase()` para obtener el valor de n .
- `void CalculaPrimeros(int tope)` para que calcule los tope primeros elementos de la sucesión.
- `int TotalCalculados()` que devuelva cuántos elementos hay actualmente almacenados (el valor tope del método anterior)
- `int k_esimo(int k)` para que devuelva el elemento k -ésimo de la sucesión.

Escriba un programa que lea los valores de dos enteros, n y k y calcule, almacene y muestre por pantalla los k primeros términos de la sucesión de Fibonacci de orden n :

```
.....
Fibonacci fibonacci(n);

fibonacci.CalculaPrimeros(k);
tope = fibonacci.TotalCalculados();    // tope = k

for (int i=0; i<tope; i++)
    cout << fibonacci.k_esimo(i) << " ";
```

Dificultad Media.

104. [Eratóstenes] (*Examen Septiembre 2012*) La **criba de Eratóstenes** (Cirene, 276 a. C. Alejandría, 194 a. C.) es un algoritmo que permite hallar todos los números primos menores que un número natural dado n .

El procedimiento consiste en escribir todos los números naturales comprendidos entre 2 y n y *tachar* los números que *no* son primos de la siguiente manera: el primero (el 2) se declara primo y se tachan todos sus múltiplos; se busca el siguiente número entero que no ha sido tachado, se declara primo y se procede a tachar todos sus múltiplos, y así sucesivamente. El proceso para cuando el cuadrado del número entero es mayor o igual que el valor de n .

El programa debe definir una clase llamada `Eratostenes` que contendrá:

- Como dato miembro debe declarar un vector privado `primos` tal que en la componente k se almacenará el primo k -ésimo ($[2, 3, 5, 7, \dots]$). El cómputo de los primos se hará en el siguiente método.
- El método `void CalculaHasta(int n)` calcula los primos menores que n . Cuando se ejecute el método, se calcularán todos los primos menores que n , según el método de Eratóstenes descrito anteriormente. Para realizar esta tarea, tendrá que definir un vector local al método con todos los números menores que n y procederá a *tachar* los no primos según el algoritmo de Eratostenes. Los números no tachados serán los primos y serán los que almacene en el dato miembro `primos`.
- El método `int TotalCalculados()` devuelva cuántos primos hay actualmente almacenados.
- `int k_esimo(int k)` para que devuelva el k -ésimo primo.

El programa principal quedaría de la forma:

```
Eratostenes primos;
int n = 100; int num_primos;

primos.CalculaHasta(n);
num_primos = primos.TotalCalculados();

for (int i=0; i<num_primos; i++)
    cout << primos.k_esimo(i) << " ";
```

Dificultad Media.

105. [Palabras en una frase] (*Examen Septiembre Doble Grado 2013*) Defina una clase `Frase` para almacenar un conjunto de caracteres (similar a la clase `SecuenciaCaracteres`). Defina un método para localizar la k -ésima palabra.

Una palabra es toda secuencia de caracteres delimitada por espacios en blanco a izquierda y derecha. La primera palabra no tiene por qué tener espacios a su izquierda y la última no tiene por qué tener espacios a su derecha. Puede haber varios caracteres en blanco consecutivos.

Si k es mayor que el número de palabras, se considera que no existe tal palabra.

Por ejemplo, si la frase es {' ', ' ', 'h', 'i', ' ', ' ', 'b', 'i', ' '}. Si $k = 1$, la posición es 2. Si $k = 2$ la posición es 6. Si $k = 3$ la posición es -1.

Si la frase fuese {'h', 'i', ' ', 'b', 'i', ' '}, entonces si $k = 1$, la posición es 0. Si $k = 2$ la posición es 3. Si $k = 3$ la posición es -1.

Dificultad Media.

106. [Palabras en una frase -continuación-] Sobre el ejercicio 105, añade los siguientes métodos:

- void EliminaBlancosIniciales() para borrar todos los blancos iniciales.
- void EliminaBlancosFinales() para borrar todos los blancos finales.
- int NumeroPalabras() que indique cuántas palabras hay en la frase.
- void BorraPalabra(int k_esima) para que borre la palabra k -ésima.
- void MoverPalabraFinal(int k_esima) para desplazar la palabra k -ésima al final de la frase.

Dificultad Media.

107. [Búsqueda por interpolación] (*Examen Prácticas Septiembre 2016*) Implemente la **Búsqueda por Interpolación** en la clase SecuenciaCaracteres. El método busca un valor buscado entre las posiciones izda y dcha y recuerda a la *búsqueda binaria* porque requiere que el vector en el que se va a realizar la búsqueda esté ordenado y en cada consulta sin éxito se descarta una parte del vector para la siguiente búsqueda.

La diferencia fundamental con la búsqueda binaria es la manera en que se calcula el elemento del vector que sirve de referencia en cada consulta (que ocupa la posición pos). Ya no es el que ocupa la posición central del subvector en el que se efectúa la búsqueda (el delimitado únicamente por izda y dcha), sino que depende también del contenido de esas casillas, de manera que pos será más cercana a dcha si buscado es más cercano a $v[dcha]$ y más cercana a izda si buscado es más cercano a $v[izda]$. En definitiva, se cumple la relación:

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