

FUNDAMENTOS DE PROGRAMACIÓN

Grado en Ingeniería Informática

GRUPO A



Guión de Prácticas

Curso 2013/2014

Autor: **Juan Carlos Cubero** (JC.Cubero@decsai.ugr.es)

Adaptación al grupo A: **Francisco José Cortijo Bon** (cb@decsai.ugr.es)

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

Aristóteles



"In theory, there is no difference between theory and practice. But, in practice, there is".

Jan L. A. van de Snepscheut



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



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



Sobre el guión de prácticas

Este guión de prácticas contiene las actividades a realizar por el alumno, especificando las que debe realizar de forma presencial (en las aulas de la ETS de Ingenierías Informática y de Telecomunicación) como no presencial (individual y en grupo). Todas las actividades son obligatorias (incluidas las presenciales en las aulas de prácticas), excepto las marcadas como *Actividades de Ampliación*.

El guión está dividido en **sesiones**, que corresponden a cada semana lectiva. Las actividades no presenciales de una sesión deben realizarse antes de la correspondiente sesión de prácticas presencial. La primera sesión presencial de prácticas (página 11) tendrá lugar durante la segunda semana de clases, por lo que el alumno deberá realizar con anterioridad las actividades encargadas.

Una actividad no presencial recurrente es la resolución de problemas, organizados en **Relaciones de Problemas**, que encontrará al final de este documento.

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

- El profesor propone con antelación suficiente la resolución de una serie de ejercicios de una relación de problemas, seleccionados de entre los **básicos**. Estos ejercicios deberán resolverse en la casa.

Las soluciones deberán ser entregadas suguiendo las indicaciones y plazos estrablecidos usando la plataforma `decsai.ugr.es`

- El alumno deberá defenderlos individualmente (a veces explicándolo a sus compañeros) en las sesiones de prácticas presenciales, durante la primera hora. La defensa será individual y forma parte de la nota final de la asignatura (10 % tal y como se explica en el método de evaluación de la asignatura).
- En la segunda hora, el profesor explicará estos ejercicios a toda la clase. También podrá sacar a la pizarra a algún alumno para que explique alguno de los ejercicios.
- Mientras el profesor corrige los ejercicios a un alumno en la primera hora, los alumnos desarrollarán las **actividades presenciales** de la correspondiente sesión. Cuando las hayan terminado, se pueden poner a trabajar en los problemas de la próxima sesión.

Del conjunto de problemas **básicos**, en el guión de prácticas se distinguirá entre:

1. **Obligatorios:**

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

2. **Opcionales:**

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

Encontrará también **actividades de ampliación**. Son problemas cuya solución no se verá, pero que sirven para afianzar conocimientos. El alumno debería intentar resolver por su cuenta un alto porcentaje de éstos.

Las actividades marcadas como *Seminario* han de hacerse obligatoriamente y siempre serán expuestas por algún alumno elegido aleatoriamente.

Para la realización de estas prácticas, se utilizará el entorno de programación Microsoft Visual Studio. En la página 5 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 Code::Blocks → <http://www.codeblocks.org/>

Muy importante:

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

Instalación de Visual Studio en nuestra casa

Conseguir Visual Studio e instalarlo

Conseguir Visual Studio

Puede encontrarse información sobre la última versión de Visual Studio en:

[http://msdn.microsoft.com/es-es/vstudio/default\(en-us\).aspx](http://msdn.microsoft.com/es-es/vstudio/default(en-us).aspx)

<http://www.microsoft.com/spain/visualstudio/products/2010-editions/ultimate>

El DVD con la instalación de Visual Studio 2010 Ultimate puede adquirirse en la fotocopiadora de la Escuela por unos 3 euros aproximadamente. La licencia es legal para todo alumno del departamento de Ciencias de la Computación e Inteligencia Artificial, fruto del programa MSDN Academic Alliance entre Microsoft y el departamento. Para más información sobre este programa:

<http://www.microsoft.com/spanish/msdn/academico.msp>

La principal restricción es que no se pueden desarrollar programas comerciales con esta licencia. Para más información sobre las restricciones de la licencia:

<http://msdn.microsoft.com/en-us/academic/bb250608.aspx>

La versión Ultimate disponible para los alumnos contiene muchas más herramientas de las que se necesitarán en la asignatura. Si se quiere, puede descargarse directamente desde Internet la versión *Express* de Visual Studio. Esta versión también contiene todas las herramientas y opciones necesarias en esta asignatura, y es la que se usará en las aulas de prácticas:

<http://www.microsoft.com/express/Downloads/>

Para descargas de versiones anteriores, consultad:

<http://msdn.microsoft.com/es-es/express/aa975050.aspx>

Los requisitos del sistema, para la instalación de Visual Studio Express se encuentran en

<http://www.microsoft.com/express/Support/>

y para Visual Studio 2010 en:

<http://www.microsoft.com/spain/visualstudio/products/2010-editions/ultimate>

Instalar Visual Studio

Si se usa el DVD de la fotocopiadora con Visual Studio 2010 Ultimate, se puede instalar la versión completa que incluye todos los lenguajes de Visual Studio como Visual Basic, C#, etc. Al introducir el DVD se lanzará automáticamente el programa de instalación. Selec-

cionamos *Instalación Personalizada*. Por ahora, instalamos únicamente la parte correspondiente al lenguaje C++ (posteriormente, podemos instalar el resto de lenguajes)

En el caso de que instalemos la versión Visual Studio C++ Express, no se instalarán el resto de lenguajes (que, obviamente, no son necesarios para esta asignatura)

Configuración

Preparar y acceder a la consola del sistema

La consola de Windows (la ventana con fondo negro que aparece al ejecutar el comando `cmd.exe`, o bien la que sale al ejecutar un proyecto de consola de Visual Studio) no está preparada por defecto para mostrar adecuadamente caracteres latinos como los acentos. Por ejemplo, al ejecutar la sentencia de C++

```
cout << "Atención"
```

saldrá en la consola un mensaje en la forma

```
Atenci5/4n
```

Para que podamos ver correctamente dichos caracteres, debemos seguir los siguientes pasos:

1. Cambiar la fuente de la consola a una que acepte caracteres Unicode. En la versión de XP de las aulas ya se ha realizado dicho cambio. En nuestra casa, tendremos que hacer lo siguiente:

```
Inicio -> Ejecutar -> cmd
```

Una vez que se muestre la consola, hacemos click con la derecha y seleccionamos *Predeterminados*. Seleccionamos la fuente *Lucida Console* y aceptamos.

2. Debemos cargar la página de códigos correspondiente al alfabeto latino. Para ello, tenemos dos alternativas:

- a) Si queremos que la consola siempre cargue la tabla de caracteres latinos, debemos modificar el registro de Windows. Lo abrimos desde

```
Inicio->Ejecutar->regedit
```

Nos situamos en la clave

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\  
\Control\Nls\CodePage
```

y cambiamos el valor que hubiese dentro de *OEMCP* y *ACP* por el de 1252. Esta es

la forma recomendada y la que se ha usado en las aulas de prácticas. Requiere reiniciar el ordenador.

Muy Importante: Si se usa otra tabla (distinta a 1252), el sistema operativo podría incluso no arrancar.

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

```
chcp 1252
```

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

```
system("chcp 1252");
```

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

Configuración del entorno

Se recomienda que se usen las mismas opciones de configuración del entorno que las establecidas en las aulas de prácticas. Para ello, deben darse los siguientes pasos.

En el fichero `fp.zip` (carpeta PRACTICAS, sección Material de Asignaturas en `decsai.ugr.es`) se encuentra la carpeta `Visual Studio\Settings`, y dentro de ésta, el fichero `SettingsAulasPracticas_VS2010.vssettings.vssettings`

Copiadlo localmente, preferiblemente dentro de la carpeta

```
Mis Documentos\Visual Studio 2010\Settings:
```

Abrid Visual Studio. Seleccionad

```
Herramientas -> Importar y exportar configuraciones ->  
Importar la configuración de entorno seleccionada ->  
Siguiente
```

Si se quiere guardar las configuraciones actuales, elegid la primera opción. Si no se ha trabajado anteriormente con Visual Studio, podemos elegir directamente la segunda opción:

```
No, sólo importar la nueva configuración,  
reemplazando la configuración actual  
-> Siguiente  
-> Examinar
```

Seleccionamos las opciones guardadas anteriormente en

Mis Documentos\Visual Studio 2010\Settings\
\SettingsAulasPracticas.vssettings:

y hacemos click en Siguiente. Aparecen algunas opciones marcadas con un símbolo de admiración, indicando que no se van a importar (pues pueden contener información privada). Aceptamos y terminamos.

Cosas a tener en cuenta:

- A partir de ahora, si realizamos algún cambio en el entorno o en las opciones de configuración (Herramientas->Opciones), éstas se guardan automáticamente en el archivo

Mis Documentos\Visual Studio 2010\
\Settings\CurrentSettings.vssettings

Es decir, el archivo SettingsAulasPracticas.vssettings no se modifica con los cambios que efectuemos en las opciones del entorno.

- Si queremos exportar nuestra configuración para utilizarla en otro sitio, basta con seleccionar

Herramientas -> Importar y exportar configuraciones ->
Exportar la configuración de entorno seleccionada

y seleccionad el nombre del fichero .vssettings en el que queremos guardar la configuración.

La plantilla _fp_2010

En el entorno Visual Studio de las aulas de prácticas, se ha incluido una *plantilla* con las opciones de configuración de compilación más usuales así como un esqueleto del programa principal que incluye la cabecera que usaremos siempre en nuestros programas.

Para instalar esta plantilla en nuestros ordenadores personales, descargad el fichero fp.zip (carpeta PRACTICAS, sección Material de Asignaturas en decsai.ugr.es), y dentro está la carpeta Visual Studio\PlantillaFP.

Seguid las instrucciones que se encuentran en el fichero Instrucciones.txt (simplemente habrá que cambiar una línea de texto de un fichero y copiar unos ficheros).

Para crear un proyecto nuevo en Visual Studio basado en esta plantilla, basta abrir Visual Studio y seleccionar:

Archivo -> Nuevo Proyecto -> Asistentes -> _fp_2010

Seleccionamos una ubicación y tecleamos un nombre para el proyecto.

Otros recursos

Dentro del fichero `fp.zip` (carpeta PRACTICAS, sección Material de Asignaturas en `decsai.ugr.es`) se encuentra la carpeta `Visual Studio\OtrosRecursos`, con los siguientes ficheros:

- `StepOver_VS_2010.reg`. Es un fichero que añade claves al registro de Windows para que el depurador de Visual Studio no entre a depurar recursos usuales como `cout`. Para ejecutarlo, basta hacer doble click sobre él (como Administrador del Sistema)
- `_limpiar.bat` y `_limpiar.exe`. El primero es una macro que llama al segundo programa. Ambos ficheros deben estar en el directorio del cual cuelgan nuestras carpetas locales de proyectos de Visual Studio. La macro borra todos los ficheros auxiliares como los ejecutables, los que contienen información de depuración, etc. En nuestra casa ejecutaremos la macro cuando nos estemos quedando sin espacio en el disco duro y antes de subir a turing los proyectos soluciones de las distintas sesiones prácticas.

Tabla resumen de accesos directos usados en Visual Studio

Ctrl-F7	Compilar
Alt-Ctrl-F7	Generar ejecutable
F5	Ejecutar programa (con depuración)
F10	Ejecución paso a paso (sin entrar en las funciones)
F11	Ejecución paso a paso (entrando en las funciones)
Shift-F11	Salir de la ejecución paso a paso de la función
Shift-F5	Detiene ejecución paso a paso
Ctrl-F5	Ejecutar programa (sin depuración)
Ctrl-Barra espaciadora	Autocompleta el código
F1	Ayuda
F9	Quita o pone un punto de interrupción
Ctrl-F10	Ejecuta instrucciones hasta el cursor y empieza depuración
Ctrl-K, Ctrl-F	Aplica formato al texto seleccionado
Ctrl-K, Ctrl-D	Aplica formato al documento
Ctrl -	Navega por el texto hacia atrás

Sesión 1

► **Actividades a realizar en casa**

Actividad: Conseguir login y password.

Antes de empezar la primera sesión de prácticas el alumno debe registrarse electrónicamente como alumno de la Universidad, tal y como se indica en

<http://csirc.ugr.es/informatica/cuentasinstitucionales/AccesoIdentificado.html>

De esta forma, obtendremos un login y un password que habrá que introducir al arrancar los ordenadores en las aulas de prácticas. La cuenta tarda 48 horas en activarse, por lo que el registro debe realizarse al menos dos días antes de la primera sesión de prácticas.

Para obtener una dirección de correo electrónico de la Universidad, hay que seguir las instrucciones detalladas en

<http://csirc.ugr.es/informatica/correoelectronico/SolicitarCuentaCorreo.html>

Actividad: Instalación de Visual Studio.

Durante la primera semana de clase, el alumno debería instalar en su casa el compilador Visual Studio. Consultad la sección de Instalación (página 5) de este guión.

Actividades de Ampliación

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

<http://loro.sourceforge.net/notes/21-dias.html>

sobre la dificultad del aprendizaje de una disciplina como la Programación.



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

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

El Entorno de Programación. Compilación de Programas

Arranque del Sistema Operativo

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

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

El alumno deberá crear el directorio (U:\FP). Si durante la sesión se requiere abrir algún fichero, éste puede encontrarse en la carpeta

H:\CCIA\Grado_FP\

También están accesibles en la web a través de la página web de la asignatura (carpeta PRACTICAS, sección Material de Asignaturas en decsai.ugr.es)

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

Nota. En el escritorio de XP, se encuentra un acceso directo a la carpeta H:\CCIA\Grado_FP.

El primer programa

Copiando el código fuente

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

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

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

`U:\FP\I_Pitagoras`

y observad los ficheros que contiene. Cada vez que creamos un programa desde Visual Studio se creará automáticamente una carpeta (en este caso `I_Pitagoras`) con distintos ficheros necesarios para el compilador Visual Studio. Por ejemplo, podemos encontrar los ficheros

```
I_Pitagoras.vcproj  
I_Pitagoras.sln  
I_Pitagoras.cpp
```

Por ahora, lo único que nos interesa saber es lo siguiente:

- Para abrir directamente Visual Studio, basta con hacer doble click sobre los ficheros `.vcproj` o `.sln`. Al hacerlo, automáticamente se abrirá Visual Studio y cargará el proyecto que, entre otras cosas, contiene el fichero de código fuente con extensión `.cpp`. Para abrirlo, haremos doble click sobre él desde el Explorador de Soluciones de Visual Studio.

Importante: No debe hacerse doble click desde el Explorador de Windows sobre el fichero con extensión `.cpp`, ya que, en dicho caso, Visual Studio lo abriría pero no podríamos compilarlo al no encontrarse dentro de un proyecto.

Para poder compilar un fichero, éste debe estar incluido en un proyecto



- El fichero realmente interesante es el que tiene la extensión `cpp`. En nuestro caso es `I_Pitagoras.cpp`. Estos serán los ficheros sobre los que escribiremos el código de nuestros programas. Este fichero se abrirá automáticamente en el editor de Visual Studio, después de que hallamos cargado el correspondiente fichero `I_Pitagoras.sln`

Haced doble click sobre el fichero `I_Pitagoras.sln`. Debe aparecer una ventana como la de la figura 1 (el sitio exacto en el que se muestran las distintas ventanas del entorno puede cambiar):

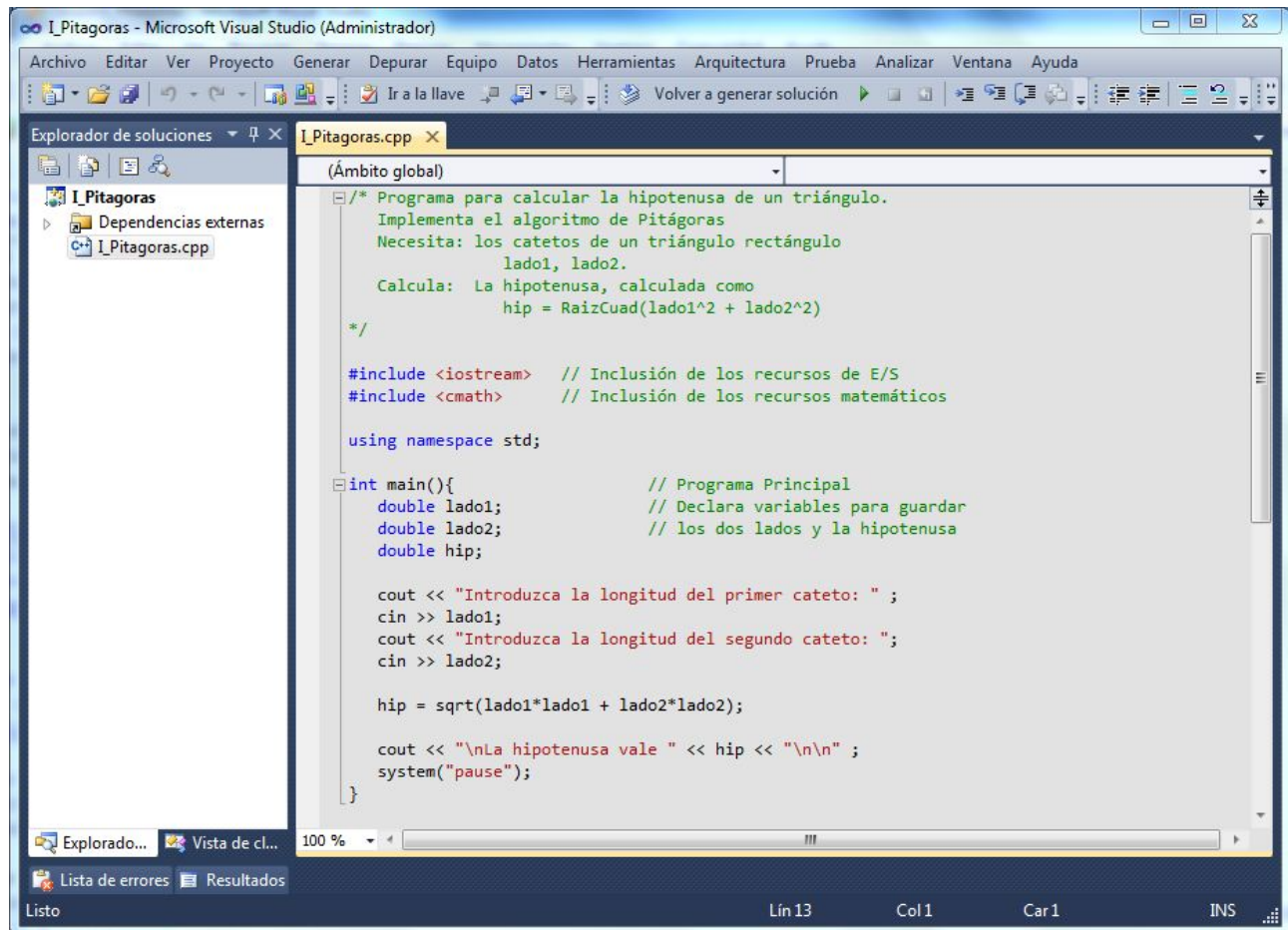


Figura 1: Programa que implementa el algoritmo de Pitágoras

Algunas consideraciones con respecto a la escritura de código en C++ (ver figura 2)

- Es bueno que, desde el principio se incluyan comentarios indicando el objetivo del programa y resaltando los aspectos más importantes de la implementación.
- Es muy importante una correcta tabulación de los programas. Por ahora, incluiremos todas las sentencias del programa principal con una tabulación. Sólo es necesario incluir la primera; el resto las pone automáticamente el entorno, al pasar a la siguiente línea.
- Para facilitar la lectura del código fuente, se deben usar espacios en blanco para separar las variables en la línea en la que van declaradas, así como antes y después del símbolo = en una sentencia de asignación. Dejad también un espacio en blanco antes y después de << y >> en las sentencias que contienen una llamada a cout y cin respectivamente.

```
int main(){
    double lado1;
    double lado2;
    double hip;

    cout << "Introduzca la longitud del primer cateto: " ;
    cin >> lado1;
    cout << "Introduzca la longitud del segundo cateto: ";
    cin >> lado2;

    hip = sqrt(lado1*lado1 - lado2*lado2);

    cout << "\nLa hipotenusa vale " << hip << "\n\n" ;
    system("pause");
}
```

Declaraciones

Entradas datos

Computos

Salida Resultados

Comentarios separados visualmente del código

espacios en blanco

Figura 2: Escritura de código

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



Observad la última sentencia

```
system("pause");
```

Veamos para qué sirve. Cuando el programa se ejecuta desde el entorno del Visual Studio, automáticamente se abre una ventana de consola para ofrecer resultados y leer datos. Cuando el programa termina de funcionar, automáticamente se cierra esta ventana, por lo que si había algún mensaje escrito en ella, no da tiempo a leerlo.


Para resolver este problema, ejecutamos el comando `pause` de la consola del Sistema Operativo. Este comando hace que se muestre el mensaje en pantalla

Pulse cualquier tecla para continuar

La ejecución del programa se detiene y cuando el usuario pulsa cualquier tecla, prosigue la ejecución.

La forma de ejecutar un comando cualquiera de la consola, desde dentro de un programa de C++, es a través del recurso `system`, pasándole como parámetro una cadena de caracteres con el comando que se quiere ejecutar. En nuestro caso sería, tal y como aparece arriba, `system("pause");`.


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 `Ctrl-F7`, 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 esquina inferior izquierda del entorno debe aparecer un mensaje del tipo:

Generación satisfactoria

Una vez compilado el programa, pasamos a generar el ejecutable, `I_Pitagoras.exe`. Para ello, o bien pulsamos `Alt-Ctrl-F7` o bien sobre el icono de la barra de herramientas que contiene el texto `Volver a generar solución`. Finalmente, para ejecutar el programa pulsamos pulsamos `F5` o sobre el icono . Si se quiere ejecutar directamente, basta pulsar `F5`, sin dar los dos primeros pasos de compilación y generación.

Debe aparecer una ventana de comandos del Sistema, en la que se estará ejecutando el programa. La ejecución del programa se detendrá en aquellos puntos del mismo donde

se requiera la interacción del usuario para poder proseguir, es decir, en las operaciones de entrada de datos a través del dispositivo estándar de entrada. En este ejemplo, sería en las dos operaciones `cin`. En el resto de los casos, la ejecución del programa continuará hasta el final. La introducción de datos mediante la sentencia `cin` se hace siempre de la misma manera; primero se introduce el valor que se desee y al terminar se pulsa la tecla RETURN. Introducid ahora los valores pedidos en el ejemplo de Pitágoras y comprobad la respuesta del programa.

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

1. Cerramos Visual Studio
2. Abrid una ventana de Mi PC.
3. Situarse en la carpeta dónde se guardó el código fuente.
4. Haced doble click sobre el fichero `I_Pitagoras.exe`.

Nota. En la instalación por defecto de Visual Studio que hagamos en nuestra casa, los ficheros de salida e intermedios necesarios por Visual Studio se almacenan en la misma carpeta del proyecto.

Prueba del programa

Uno podía pensar que una vez que consigo un fichero ejecutable a partir de mi código fuente, el problema está terminado. Sin embargo esto no es así. Tras el proceso de compilado se requiere una fase de prueba. Dicha fase intenta probar que el algoritmo planteado resuelve el problema propuesto. Para llevar a cabo esta fase, es necesario ejecutar el programa y verificar que los resultados que obtiene son los esperados.

Ahora que podemos ver el resultado obtenido por el programa implementado, verifiquemos mediante el siguiente conjunto de pruebas que el programa funciona de forma correcta.

lado1	lado2	hip
3	4	5
1	5	5.099
2.7	4.3	5.077
1.25	2.75	3.02

Una vez que el algoritmo supera la fase de prueba, podemos considerar que se ha concluido con la fase inicial del desarrollo del software.

Introducción a la corrección de errores

Los errores de compilación

Ya hemos visto los pasos necesarios para construir un fichero ejecutable a partir del código fuente. El paso central de este proceso era la fase de compilación. En esta parte de este guión de prácticas aprenderemos a corregir los errores más comunes que impiden una compilación exitosa del fichero fuente.

Cargad el proyecto `I_Pitagoras.sln` (no olvidad que desde el Sistema Operativo hay que seleccionar el fichero con extensión `vcproj` o `sln`). En la ventana del *Explorador de Soluciones*, haced doble click sobre `I_Pitagoras.cpp` para poder editar el fichero. Quitadle una 'u' a alguna aparición de `cout`. Intentad compilar. Podemos observar que la compilación no se ha realizado con éxito. Cuando esto sucede, en la parte inferior de la ventana principal, en la pestaña *Lista de errores*, aparecen los errores que se han encontrado. Aparece una descripción del error, así como otra información, como el número de línea en la que se produjo. Los pasos que debemos seguir para la corrección son los siguientes:

1. Ir a la primera fila de la lista de errores.
2. Leer el mensaje de error e **intentar entenderlo**.
3. Hacer doble click sobre esa fila con el ratón. Esto nos posiciona sobre la línea en el fichero fuente donde el compilador detectó el error.
4. Comprobar la sintaxis de la sentencia que aparece en esa línea. Si se detecta el error, corregirlo. Si no se detecta el error mirar en la línea anterior, comprobar la sintaxis y repetir el proceso hasta encontrar el error.
5. Después de corregir el posible error, guardamos de nuevo el archivo y volvemos a compilar. Esto lo hacemos aunque aparezcan más errores en la ventana. La razón es que es posible que el resto de los errores sean consecuencia del primer error.
6. Si después de corregir el error aparecen nuevos errores, volver a repetir el proceso desde el paso 1.

A veces, el compilador no indica la línea exacta en la que se produce el error, sino alguna posterior. Para comprobarlo, haced lo siguiente:

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

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

1. Cambiad algún punto y coma por cualquier otro símbolo
2. Cambiad `double` por `dpuble`
3. Cambiad la línea `using namespace std;` por `using namespace STD;`
4. Poned en lugar de `iostream`, el nombre `iotream`.
5. Borrard alguno de los paréntesis de la declaración de la función `main`
6. Introducid algún identificador incorrecto, como por ejemplo `cour`
7. Usad una variable no declarada. Por ejemplo, en la definición de variables cambiad el nombre a la variable `lado1` por el identificador `lado11`.
8. Borrard alguna de las dobles comillas en una constante de cadena de caracteres, tanto las comillas iniciales como las finales.
9. Borrard alguna de las llaves que delimitan el inicio y final del programa.
10. Borrard la línea `using namespace std;` (basta con comentarla con `//`)
11. Cambiad un comentario iniciado con `//`, cambiando las barras anteriores por las siguientes `\\`
12. Cambiad la aparición de `<<` en `cout` por las flechas cambiadas, es decir, `>>`. Haced lo mismo con `cin`.
13. Suprimid todo el `main`. No hace falta borrar el código, basta con comentarlo.
14. Aunque no sea un error, comentad la última línea que contiene `system("pause");` para comprobar que el programa se ejecuta correctamente, pero no da tiempo a ver los resultados en la consola.

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

Los errores lógicos y en tiempo de ejecución

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

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

- Cambiad la sentencia
`sqrt(lado1*lado1 + lado2*lado2)` por:
`sqrt(lado1*lado2 + lado2*lado2)`
Ejecutad introduciendo los lados 2 y 3. El resultado no es correcto, pero no se produce ningún error de compilación ni en ejecución. Es un error lógico.
- Para mostrar un error de ejecución, declarad tres variables ENTERAS (tipo `int`) `resultado`, `numerador` y `denominador`. Asignadle cero a `denominador` y 7 a `numerador`. Asignadle a `resultado` la división de `numerador` entre `denominador`. Imprimid el resultado. Al ejecutar el programa, se produce una excepción o error de ejecución al intentar dividir un entero entre cero.

Creación de un programa nuevo

En esta sección vamos a empezar a crear nuestros propios programas desde Visual Studio. El primer ejemplo que vamos a implementar corresponde al ejercicio 1 sobre la Ley de Ohm, de la relación de problemas I.

Para crear un programa nuevo, abrimos Visual Studio y elegimos

Archivo->Nuevo Proyecto

Si tenemos instalados varios lenguajes (Visual Basic, C#, etc), elegimos C++. Nos aparecen distintos tipos de proyectos, en función de lo que vayamos a crear (una aplicación en Consola, un formulario de Windows, etc). Elegimos Proyecto vacío. En la parte inferior de la ventana, hacemos click sobre Examinar para elegir la carpeta donde colgará el directorio de nuestra aplicación. En nuestro caso, será `U:\FP`. Finalmente elegimos un nombre para nuestra aplicación. Teclead `I_Voltaje` y pulsad sobre Aceptar (ver figura 3)

Se nos abrirá una ventana con el Explorador de Soluciones.

Sobre la carpeta Archivos de código fuente pulsamos click con la derecha y seleccionamos Agregar->Nuevo elemento. Seleccionamos Archivo C++ (.cpp) y le damos un nombre. Podemos darle el mismo nombre que el del proyecto, `I_Voltaje` (ver figura 4)

Llegado a este paso, debemos tener el entorno de Visual Studio, con varias ventanas como las de la figura 5 (la situación exacta de las ventanas puede variar)

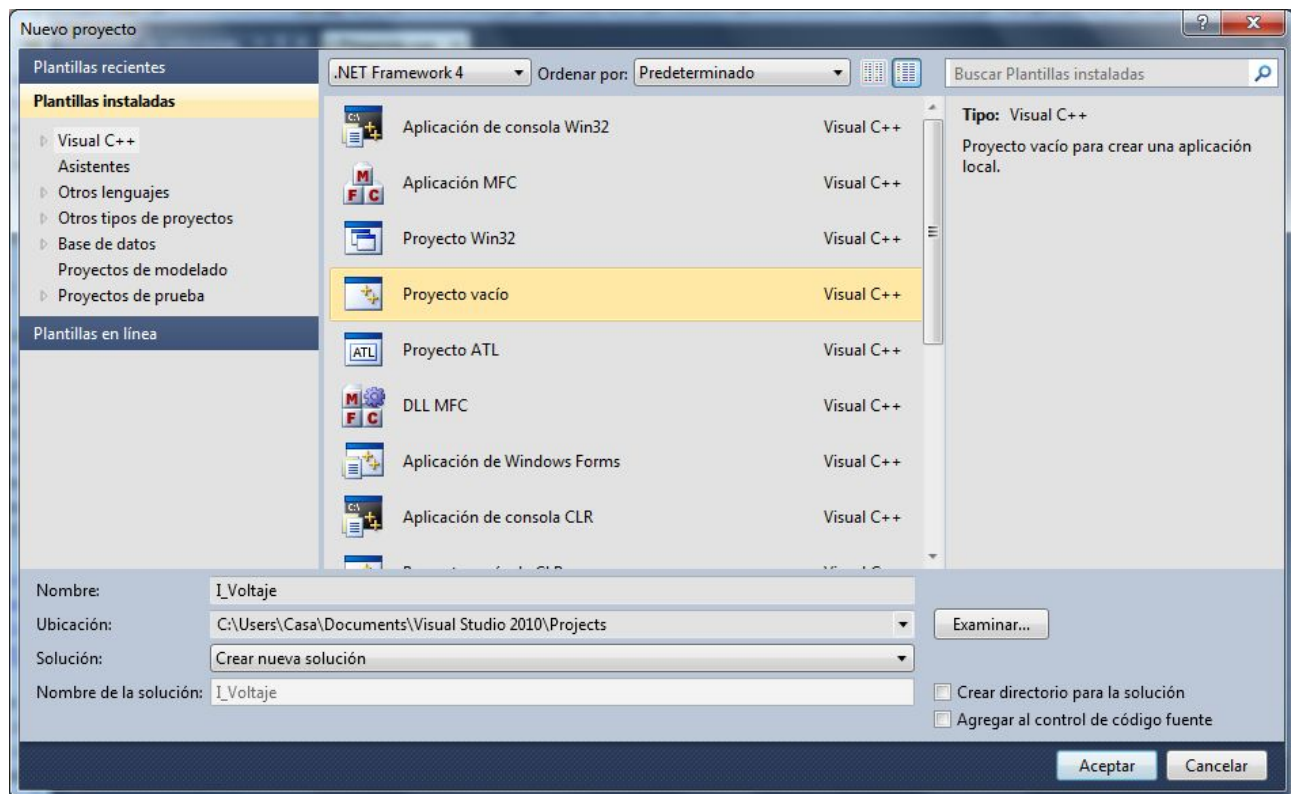


Figura 3: Creación de un proyecto nuevo

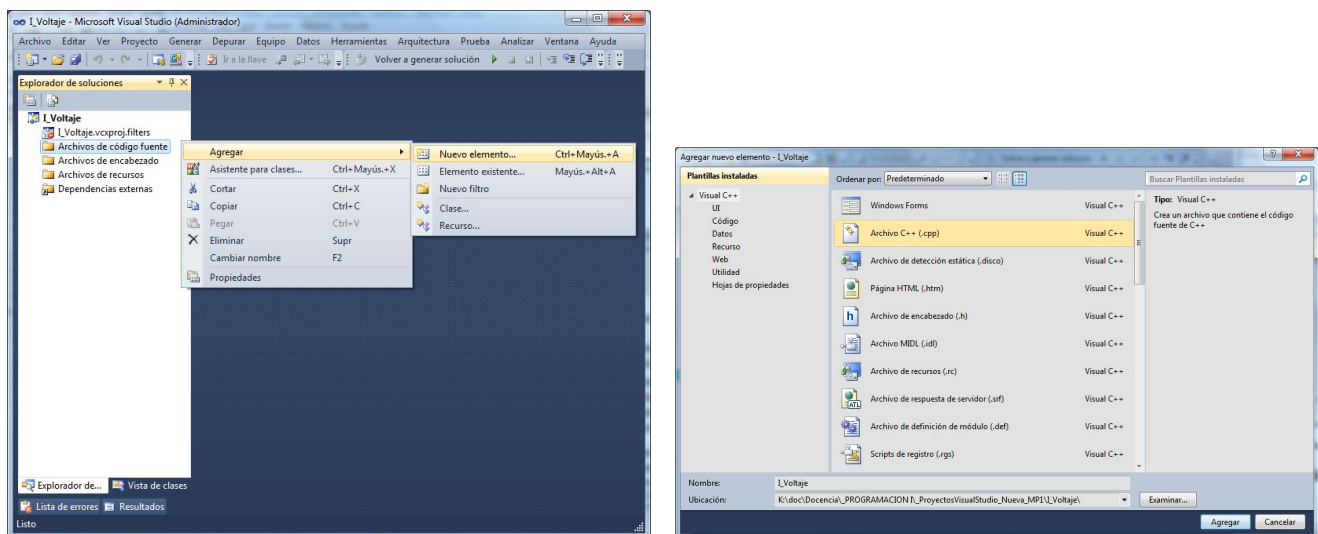


Figura 4: Añadiendo un fichero de código vacío

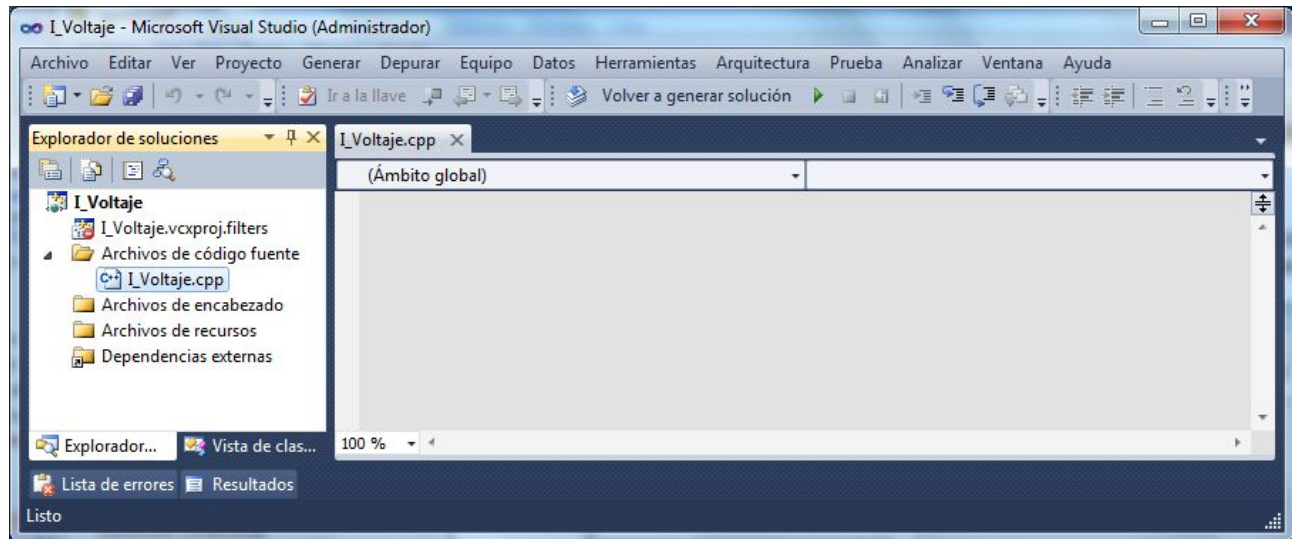


Figura 5: Entorno de Visual Studio

Ya estamos en condiciones de resolver el problema pedido. Escribimos el código en la ventana correspondiente a la pestaña `I_Voltaje.cpp`. Habrá que leer desde teclado los valores de intensidad y resistencia y el programa imprimirá en pantalla el voltaje correspondiente. Recordad que compilamos con `Ctrl-F7` y ejecutamos con `F5`.

Nota. Cuando tenemos varias variables en el código, podemos empezar a escribir el nombre de alguna de ellas y antes de terminar, pulsar `Ctrl-Barra espaciadora`. La ayuda nos mostrará los identificadores disponibles que empiecen por las letras tecleadas.

A partir de ahora, cada uno de los ejercicios irá en un proyecto nuevo. Implementad los ejercicios que se habían mandado para esta sesión en este orden: 2, 3, 4, 5, 6. Usad en esta sesión el tipo de proyecto "Proyecto Vacío".

Sesión 2

► Actividades a realizar en casa

Actividad: Resolución de problemas.

Resolved los ejercicios siguientes de la Relación de Problemas I (página **RP-I.1**). Para cada uno de ellos, habrá que crear el correspondiente proyecto. Usad el tipo de proyecto "Proyecto Vacío".

Recordad que antes del inicio de esta sesión en el aula de ordenadores hay que entregar las soluciones mediante `decsai.ugr.es`.

- *Obligatorios:*
 - 1** (Voltaje)
 - 2** (Interés bancario)
 - 3** (Intercambiar dos variables)
 - 4** (Circunferencia)
 - 5** (Gaussiana)
 - 6** (Fabricante)
- *Opcionales:*
 - 7** (Media aritmética)

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

Durante esta sesión de prácticas, el profesor irá corrigiendo individualmente (a algunos alumnos elegidos aleatoriamente) los ejercicios indicados anteriormente. Mientras tanto, el resto de alumnos deben seguir las instrucciones indicadas en los siguientes apartados. Finalmente, el profesor explicará los ejercicios.

Configuración del proyecto y gestión de errores

Los proyectos en Microsoft Visual C++ pueden ser generados con muchas opciones. Por ejemplo, se puede añadir al fichero ejecutable información de depuración, para poder realizar un seguimiento de las instrucciones ejecutadas, o que detecte los problemas para exportar el programa a una plataforma de 64 bits, etc. Estas opciones están disponibles desde el Explorador de Soluciones, pinchando con la izquierda el proyecto y seleccionando **Propiedades**, o bien directamente desde el menú **Proyecto->Propiedades**. No vamos a ver ninguna de estas opciones, aunque sí debemos saber cómo influyen en el desarrollo de un programa. Normalmente, los programas se desarrollan en dos fases:

1. En una primera fase, se usa un entorno optimizado de depuración. Esto supone una recarga computacional importante, pero es muy adecuada para detectar errores de programación, como por ejemplo, el usar una variable no asignada previamente. Cuando se trabaja en esta fase, hay que seleccionar

Proyecto -> Propiedades

y pinchad sobre **Administrador de configuración** en la esquina superior derecha. Dentro de **Configuración de soluciones activas**, seleccionad **Debug**. Esta es la configuración adecuada en esta primera fase. Para probarlo, copiad la carpeta **I_Voltaje** del directorio de la asignatura a la unidad local **U:\FP**. Abrid el proyecto y comentad la sentencia en la que se realiza la lectura de la intensidad

```
// cin >> intensidad;
```

Cuando se llegue a ejecutar la sentencia

```
voltaje = intensidad * resistencia;
```

el valor de **intensidad** será basura. Si compilamos el programa con **Ctrl-F7** podemos apreciar que se genera una advertencia en la lista de errores, indicando que se está usando la variable **intensidad** sin haberle asignado previamente un valor. Aún así, podemos generar el fichero **.exe** y ejecutarlo. Hacedlo, para comprobar el error que se obtiene en tiempo de ejecución.

Nota. La generación de información de depuración conlleva la creación de ficheros que pueden ser bastante grandes. Para evitar que dichos ficheros se almacenen en la unidad local del alumno U:\ (ya que tiene un límite de almacenamiento) en las aulas de ordenadores se ha configurado Visual Studio para que se almacenen en la carpeta:

C:\Visual Studio Output_Ejecutables

En la instalación por defecto de Visual Studio, esta carpeta se encuentra dentro del directorio del proyecto. Si en nuestra casa hemos realizado la instalación por defecto, es posible que de vez en cuando queramos eliminar dichos ficheros temporales. Para ello, copiaremos el fichero `limpiar.exe` y la macro `_limpiar.bat` que se encuentran en el archivo `fp.zip` (carpeta PRACTICAS, sección Material de Asignaturas en `decsai.ugr.es`) y los guardaremos en la carpeta de la que cuelgan nuestros proyectos de Visual Studio. Ejecutaremos la macro `_limpiar.bat` para eliminar los ficheros de depuración y ejecutables que se encuentran en las carpetas de los proyectos.

2. En una segunda fase, cuando ya hemos comprobado que el programa funciona correctamente, se procede a generar el fichero ejecutable (`.exe`) eliminando la información de depuración (entre otras cosas). Esto hará que el ejecutable obtenido sea bastante más pequeño. Para ello, hay que cambiar la configuración antes de generar el ejecutable. Volvemos a entrar en

Configuración de soluciones activas

y seleccionamos Release. Podemos hacerlo directamente desde el botón disponible en la esquina superior derecha de la barra de herramientas. Si generamos el programa bajo esta última configuración, veremos que al ejecutarlo no se produce un error durante la ejecución. Esto es debido, como ya hemos comentado, a que el compilador ya no *pierde el tiempo* comprobando estos posibles errores de programación. Por contra, obtendremos un error lógico ya que la variable `intensidad` contiene basura y al multiplicarla por la variable `resistencia` el resultado será basura.

A lo largo de la asignatura, usaremos siempre la configuración Debug, por lo que algunos errores de programación, como el uso de variables no asignadas previamente, serán detectados por el compilador. Pero debemos tener claro que en la configuración Release, o bien en la compilación realizada con otros compiladores distintos a Visual Studio, no se producirá ningún error durante la ejecución sino un lamentable error lógico.

En el entorno Visual Studio de las aulas de prácticas, se ha incluido una *plantilla* con las opciones de configuración de compilación más usuales (tanto para la configuración Debug como Release) así como un esqueleto del programa principal que incluye la cabecera que usaremos siempre en nuestros programas. Para crear un programa nuevo basándonos en esta plantilla, basta hacer lo siguiente:

Archivo -> Nuevo Proyecto -> `_fp_2010`

Seleccionamos una ubicación (usualmente `U:\FP`) y tecleamos un nombre para el proyecto. A partir de esta sesión usaremos siempre la plantilla `_fp_2010` para crear nuestros proyectos.

Para instalar esta plantilla en nuestros ordenadores personales, seguid las instrucciones que se encuentran en la página [5](#)

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

[20](#) (Locomotoras)

[21](#) (Triángulo)

Sesión 3

Tipos básicos y operadores - Estructura condicional

► **Actividades a realizar en casa**

Actividad: Resolución de problemas.

Resolved los siguientes problemas. Por cada uno de ellos, habrá que crear el correspondiente proyecto. Usad el tipo de proyecto `_fp_2010`

- **Obligatorios:**

De la Relación de Problemas I (página **RP-I.1**):

- 9** (Horas, minutos y segundos)
- 13** (Pasar de mayúscula a minúscula)
- 14** (Pasar de carácter a entero)
- 16** (Expresiones lógicas)
- 22** (Pinta dígitos)

De la Relación de Problemas II (página **RP-II.1**):

- 2** (Subir ingresos a jubilados)
- 3** (Ver si dos números se dividen)
- 4** (Pasar de mayúscula a minúscula)
- 5** (Pasar de mayúscula a minúscula y viceversa)

- **Opcionales:**

- 6** (Medidas contra la crisis)
- 7** (Tarifa aérea)

Actividades de Ampliación



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

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

para una introducción básica a los conceptos, y

<http://club.telepolis.com/ildearanda/index.html>

como una referencia a las fórmulas que nos proporcionan los cálculos correspondientes.

2. Ejecutad el applet

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

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

3. Hojear la página

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

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

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

Herramientas de ayuda para la edición de código fuente


En esta sesión empezaremos a usar las estructuras condicionales. En primer lugar, crearemos el proyecto `II_Media_int_ampliado` a partir del fichero fuente `II_Media_int_ampliado.cpp` (disponible en la carpeta Actividades Aulas, componente de la carpeta PRACTICAS, sección Material de Asignaturas en decsai.ugr.es) que contiene la solución al ejercicio 1 de la relación de problemas II (página **RP-II.1**). Sobre este proyecto, probaremos las nuevas herramientas que vamos a ver a continuación.

Es muy importante poner atención a la tabulación correcta de las sentencias, tal y como se indica en las transparencias. Recordad que una tabulación incorrecta supondrá bajar puntos en la primera prueba práctica que se realizará dentro de algunas semanas. Visual Studio ofrece algunas herramientas para agilizar la correcta tabulación:

- VS tabula automáticamente el código si así se especifica en:

```
Herramientas -> Opciones -> Editor de texto  
-> C/C++ -> Tabulaciones -> Automático
```

El fichero de configuración `.settings` usado en las aulas de prácticas viene con dicha opción seleccionada.

- De forma semi-automática, podemos seleccionar un bloque de texto con el ratón y pulsar **Ctrl-K, Ctrl-F**.
- De forma manual, podemos seleccionar un bloque de texto con el ratón y pulsar sobre alguno de los iconos  para disminuir o aumentar la sangría. Las teclas de acceso directo para estas operaciones son **Shift-Tabulador** y **Tabulador** respectivamente.

Visual Studio ofrece también un recurso bastante útil para la escritura de código. Cuando nuestros programas contienen muchas líneas de código, nos gustaría *ocultar* las partes que se han comprobado que funcionan correctamente. Para ello, basta seleccionar con el ratón un bloque de texto y seleccionar con la derecha

Esquemmatización -> Ocultar selección

De esta forma conseguimos que en el margen izquierdo del editor de texto aparezca un símbolo +. Si pinchamos sobre él, se despliega mostrando el código y si pulsamos otra vez, se oculta dicha selección. Hay que destacar que el código que estamos *ocultando* sigue siendo código del programa, por lo que se compilará y ejecutará como cualquier otro código. Dicho de otra forma, por el hecho de ocultar de esta forma un trozo de código, no lo estamos incluyendo como un comentario.

Forzad los siguientes errores en tiempo de compilación, para habituarnos a los mensajes de error ofrecidos por el compilador:

- Suprimid los paréntesis de alguna de las expresiones lógicas de la sentencia **if**
- Quitad la llave abierta de la sentencia condicional (como únicamente hay una sentencia dentro del **if** no es necesario poner las llaves, pero añadimos las llaves a cualquier condicional para comprobar el error que se produce al eliminar una de ellas)
- Quitad la llave cerrada de la sentencia condicional

Depuración

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



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

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



Seguid trabajando con el proyecto `II_Media_int_ampliado`. Una vez abierto lo compilamos para que se genere la información de depuración. Dicha información se puede almacenar en distintos sitios. En la plantilla de `_fp_2010` se ha seleccionado que sea sobre un fichero con extensión `.pdb`.

Nota. Para más información, id al Explorador de Soluciones, pinchad sobre el proyecto con el click derecho y seleccionad ayuda con F1 sobre:

Proyecto -> Propiedades -> Propiedades de configuración
-> C/C++ -> General -> Formato de la información de depuración

Si en la anterior opción se eligiese Deshabilitado, no se generaría ninguna información de depuración.

La idea básica en la depuración es ir ejecutando el código línea a línea para ver posibles fallos del programa. Para eso, debemos dar los siguientes pasos:

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


Para establecer un punto de ruptura podemos mover el ratón en la parte más a la izquierda de una línea de código, y pulsar el botón izquierdo 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 pulsar **F9**. Para eliminar un punto de ruptura, se realiza la misma operación que para incluirlo, sobre la instrucción que actualmente lo tiene.

Colocad ahora un punto de ruptura sobre la línea que contiene la primera sentencia condicional `if`.

2. A continuación le pediremos al depurador que empiece a ejecutar nuestro programa pulsando como siempre **F5**. Esto provocará que se ejecuten todas las sentencias que hay justo antes del punto de ruptura. Llegado a este punto, la ejecución se pausa y se muestra en amarillo aquella línea que se va a ejecutar a continuación (en este caso, la que contiene el punto de interrupción)
3. Una vez que tenemos la línea amarilla sobre una instrucción, para proceder a su ejecución, pulsamos **F10**. Sólo se ejecuta una sentencia (línea) y se marca en amarillo la siguiente línea, en espera a que volvamos a pulsar F10. Desde la barra de herramientas, podemos pinchar sobre el icono

Ejecutad varias líneas de programa pulsando F10. Cuando se ejecute una sentencia `cin`, habrá que irse a la ventana de la consola e introducir el valor pedido, y una vez hecho, volver a Visual Studio para proseguir la ejecución con F10.

4. Para proseguir la ejecución del programa sin ir paso a paso, basta con pulsar de nuevo F5. Para detener la ejecución (y por tanto también la depuración), seleccionaríamos

Shift-F5, o bien el cuadrado de la barra de herramientas .

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.

5. Cuando lleguemos a la última línea del programa, la que contiene la llave cerrada del `main`, al pulsar F10, se abrirá una pestaña nueva sobre un fichero denominado `crt0.c`. Este fichero es utilizado internamente por Visual Studio como punto de entrada de la aplicación. Para evitar que aparezca este fichero, cuando lleguemos a la última línea del programa, pulsaremos F5 en vez de F10.

Otros recursos:



Este comando, accesible también con **Ctrl-F10**, ejecuta todas las instrucciones hasta donde se encuentre actualmente el cursor de la ventana de edición. Al llegar a él, se marca en amarillo dicha línea y se queda en espera a que iniciemos una ejecución paso a paso, o a que prosigamos la ejecución ininterrumpida con F5.

Se utiliza para saltar determinadas sentencias que no queremos revisar.

- Si pulsamos **Ctrl-F5** en vez de F5, el programa se ejecutará sin procesar la información de depuración, por lo que, por ejemplo, no se parará en ningún punto de interrupción. Además, al ejecutar el programa de este forma, también saldrá un mensaje al final con la leyenda *Pulse cualquier tecla para continuar*. Este mensaje se añadirá al que nosotros ya habíamos puesto con `system("pause")`.
- Podemos iniciar la ejecución paso a paso desde el principio, sin poner ningún punto de interrupción, pulsando directamente F10. Esto hará que la primera instrucción destacada en amarillo sea la correspondiente a `main`.

Los depuradores de programas permiten consultar los valores que tienen en cada momento algunas de las variables que se utilizan en el programa, para así intentar descubrir dónde puede estar el posible error lógico del programa. **EN PRIMER LUGAR**, debemos iniciar la depuración introduciendo un punto de interrupción y pulsando F5 (o empezando desde el principio con F10). Automáticamente aparecen las ventanas siguientes:

- **Automático:** Contiene una lista de las variables involucradas en la sentencia que se va a ejecutar, así como las involucradas en la sentencia anterior
- **Locales:** Contiene una lista de las variables definidas en el bloque actual. Por ahora, las que aparecen son las que están definidas dentro de `main`.

- **Inspección:** Contiene variables que explícitamente ha incluido el programador. Ya veremos en temas posteriores en qué situaciones puede resultar útil.

Los valores mostrados para cada variable en las anteriores ventanas, son el valor actual para cada una de ellas. Por valor actual entendemos el valor que tiene dicha variable, antes de que se ejecute la línea marcada en amarillo. El valor de cada variable se irá actualizando de forma automática conforme el programa siga su ejecución.

Además, si durante la ejecución interrumpida situamos el cursor sobre alguna variable del código escrito `.cpp`, aparecerá automáticamente una leyenda informativa con el valor actual de dicha variable. También podemos seleccionar una expresión con el ratón (como cualquier selección de texto) y se mostrará la evaluación de dicha expresión.

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

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



Sesión 4

Estructura Repetitiva. Bucles while

► **Actividades a realizar en casa**

Actividad: Resolución de problemas.

Resolved los siguientes problemas de la Relación de Problemas II (página **RP-II.1**). Por cada uno de ellos, habrá que crear el correspondiente proyecto. Usad el tipo de proyecto `_fp_2010`

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

- **Obligatorios:**
 - 9** (Divisores de un número)
 - 11** (Reinvertir capital e interés un número de años)
 - 12** (Reinvertir capital e interés hasta doblar cantidad inicial)
 - 13** (Filtro entrada mayúscula)
 - 17** (Ciclos pre ó post condición)
- **Opcionales:**
 - 10** (El mínimo de varios reales leídos desde teclado).
 - Nota:** Sobre este ejercicio se trabajará en el aula, en esta misma sesión de prácticas (*Actividades a realizar en las aulas de ordenadores*)
 - 18** (Sucursal con más ventas).
 - Nota:** Sobre este ejercicio se trabajará en el aula, en esta misma sesión de prácticas (*Actividades a realizar en las aulas de ordenadores*)
 - 19** (Progresión geométrica)

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

Redireccionando las entradas de cin

Cada vez que se ejecuta `cin`, se lee un dato desde el periférico por defecto. Nosotros lo hemos hecho desde el teclado, introduciendo un dato y a continuación un ENTER. Otra forma alternativa es introducir un dato y luego un separador (uno o varios espacios en blanco o un tabulador). Para comprobarlo, copiad localmente la carpeta `ProblemasII\II_cin` que se encuentra en el directorio de la asignatura. Observad el código del programa y ejecutadlo desde Visual Studio. Para introducir los datos pedidos (un entero y dos caracteres) separadlos por espacios en blanco. Pulsad ENTER al final (una sola vez).

Los datos anteriores se pueden poner dentro de un fichero de texto. Dentro del fichero, cada dato hay que separarlo por uno o varios espacios en blanco o tabuladores. Para poder leer los datos del fichero, basta con ejecutar el programa `.exe` desde una consola del sistema y especificar que la entrada de datos será desde un fichero a través del símbolo de redireccionamiento `<` (no ha de confundirse con el token `<<` que aparecía en una instrucción `cout` de C++) Hay que destacar que este redireccionamiento de la entrada lo estamos haciendo en la llamada al ejecutable desde la consola del sistema operativo¹. Para probarlo, seleccionad el fichero `II_cin_datos_entrada.txt` que se encuentra dentro de la carpeta `II_cin` y copiadlo dentro de Mis Documentos. Copiad también dentro de Mis Documentos, el fichero ejecutable `II_cin.exe` que se encuentra en la carpeta `II_cin\Debug`

Abrimos Mis Documentos desde el sistema operativo y seleccionamos con el click derecha del ratón "Abrir Símbolo del Sistema". Introducimos la instrucción siguiente (ver Figura 6):

```
II_cin.exe < II_cin_datos_entrada.txt
```

Ahora, cada vez que se ejecute una instrucción `cin` en el programa, se leerá un valor de los presentes en el fichero de texto.

Visual Studio también permite redireccionar la entrada de datos desde el propio entorno. Esto resulta muy útil cuando se desea leer datos de un fichero y a la misma vez depurar el programa (introducir puntos de interrupción, ejecución paso a paso, evaluación de variables, etc). Para hacerlo, seleccionad

```
Proyecto -> Propiedades de ....  
Propiedades de Configuración -> Depuración  
Argumentos del comando -> Editar ->
```

¹ También pueden leerse datos de un fichero desde dentro del propio código fuente del programa, pero esto se verá en el segundo cuatrimestre

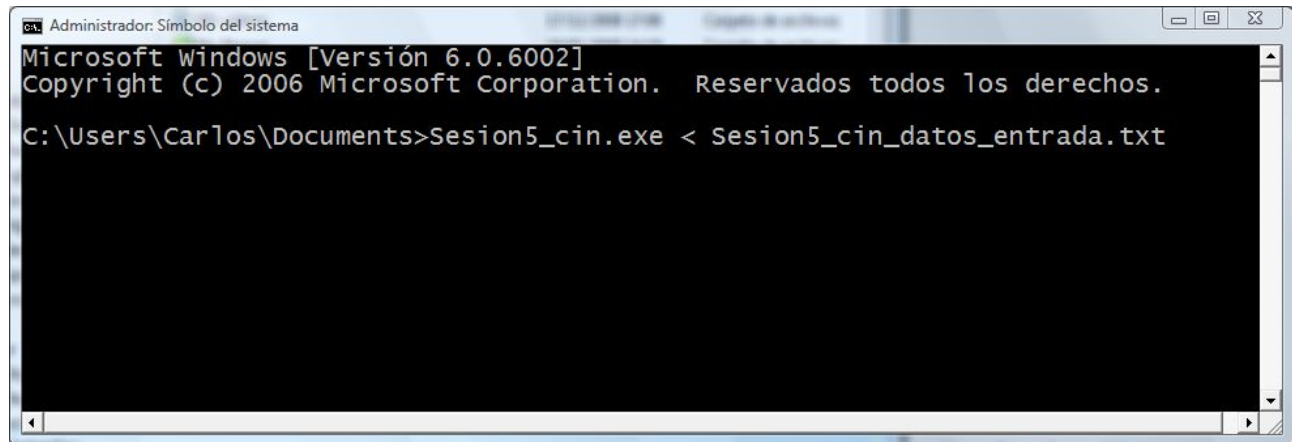


Figura 6: Redirección de la entrada desde la consola

En el cuadro de texto que aparece al seleccionar **Editar**, introduciremos el símbolo `<` seguido del nombre del fichero de texto que contiene los datos(ver Figura 7). Con el anterior ejemplo sería:

```
< II_cin_datos_entrada.txt
```

Por defecto, el entorno busca el fichero de entrada de datos en la misma carpeta que el proyecto. Si se quiere, puede indicarse otra carpeta distinta, a través de la opción siguiente:

```
Proyecto/Propiedades
  Propiedades de Configuración/Depuración
    Directorio de trabajo/
```

Cuando ejecutemos el programa, cada ejecución de `cin` leerá un dato desde el fichero indicado, saltándose previamente todos los espacios en blanco y tabuladores que hubiese previamente. Cuando llegue al final del fichero, la lectura de un valor cargaría el carácter *fin de fichero*. Cuando llega al final del programa y se ejecuta la instrucción `system("pause")` el programa se detiene y lee una tecla del dispositivo por defecto, que en este caso es el fichero. Lee la marca de fin de fichero como tecla válida y por tanto no hace la pausa que estábamos esperando. Para solucionarlo, basta con que introduzcamos un punto de ruptura al final del código fuente de nuestro programa.

Nota. En nuestra casa, o bien instalamos un programa que permita abrir una consola en el directorio actual, como por ejemplo *Open Command Prompt Shell Extension* disponible en <http://code.kliu.org/cmdopen/> o bien abrimos un símbolo del sistema (Inicio->Ejecutar->cmd) y vamos cambiando de directorio con la orden `cd`

Nota. CodeBlocks también permite, en teoría, la ejecución paso a paso cuando los datos se leen desde un fichero (desde la opción Project-Set program's arguments). Sin embargo, en la versión 10.5 esta opción no funciona correctamente bajo Windows.

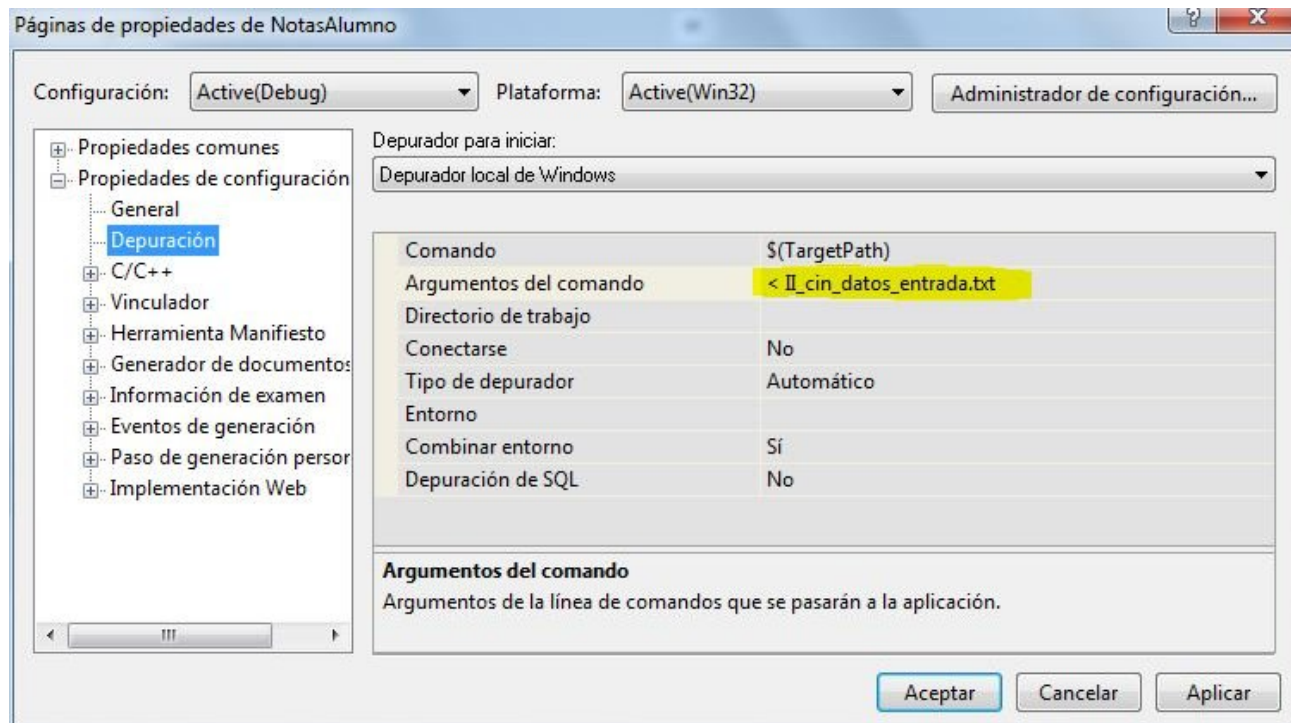


Figura 7: Redirección de la entrada desde el entorno

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

10 (El mínimo de varios reales leídos desde teclado)

18 (Sucursal con más ventas)

Ejecutar los programas leyendo los valores que van a procesar desde un fichero de texto empleando la redirección de entrada.

Utilizad las herramientas de depuración vistas en la sesión anterior para inspeccionar el valor de las variables usadas en cada iteración del bucle.

Sesión 5

Estructura Repetitiva: bucles for y bucles anidados

► **Actividades a realizar en casa**

Actividad: Resolución de problemas.

Resolved los siguientes problemas de la Relación de Problemas II (página **RP-II.1**). Por cada uno de ellos, habrá que crear el correspondiente proyecto. Usad el tipo de proyecto `_fp_2010`.

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

Durante esta sesión no hay ninguna sección "*Actividades a realizar en las aulas de ordenadores*". Durante las dos horas, los alumnos realizarán los problemas que les indique el profesor y el resto del tiempo se dedicará a la explicación y discusión sobre las soluciones de los ejercicios.

- **Obligatorios:**

- 20** (Usando un ciclo `for` reescribir los programas: a) Divisores de un número, b) Reinvertir capital e interés un número de años y c) Progresión geométrica)

- 21** (Sumatorio de la serie)

- 22** (Método RLE)

- 23** (Interés con anidamiento)

- 24 y 25** (Pintar pirámide y cuadrado)

- 26** (Pintar pirámide y cuadrado genérico)

- **Opcionales:**

- 14** (Potencia y factorial)

- 16** (Número narcisista)

- 27** (Número feliz)

.

Sesión 6

Funciones

► **Actividades a realizar en casa**

Actividad: Resolución de problemas.

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

- *Obligatorios:*
 - 2 (Máximo de 3 números reales)
 - 3 (Potencia y factorial con funciones)
 - 5 (Comprobar si una letra es minúscula y pasar a mayúscula)
 - 6 (Mayúscula a minúscula y viceversa)
 - 7 (Gaussiana)
 - 8 (Suma de serie)
- *Opcionales:*
 - 23 (Tarifa aérea)
 - 24 (Calcular CDF de una Gaussiana)

Actividades de Ampliación

Familiarizarse con las webs de referencias de funciones estándar.

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

<http://www.cppreference.com>




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

Durante la primera media hora de clase, seguid las instrucciones indicadas en el apartado siguiente *Depuración de funciones* (el otro apartado es para hacerlo en casa), mientras que el profesor corrige individualmente los ejercicios de algunos alumnos. Durante el resto de la sesión, el profesor irá llamando a los alumnos para que expliquen al resto de compañeros los ejercicios de esta sesión.

Depuración de funciones

- **Ejecución paso a paso dentro de las funciones.**

Abrid el proyecto `III_Max`, disponible en el directorio de Material de la Asignatura. Incluid un punto de interrupción en la primera línea del `main`. Ejecutad con F5. Si pulsamos F10 ejecutaremos el programa paso a paso, pero cuando llegue a la sentencia que llama a la función `Max`, el depurador no entrará en el código de dicha función. Para poder hacerlo, habrá que pulsar **F11** justo antes de la llamada a la función. Se puede alternar F10 y F11 de forma arbitraria. El icono de acceso directo a esta funcionalidad es . Probadlo.

Si pulsamos F11 justo en la llave cerrada del `main`, el depurador entrará en el código de una función especial que se encarga de ejecutar el programa. Para salirnos, o bien pulsamos F5, o bien interrumpimos la ejecución. Para no entrar en este código, cuando lleguemos a la llave cerrada del `main`, pulsaremos F10 o F5.

- **Salirse de una función.**

Si durante la depuración entramos en el código de una función, por ejemplo `Max`, y queremos ejecutar *de golpe* el resto de la función `Max` y proseguir paso a paso a partir de la línea desde la que se llamó a `Max`, pulsaremos **Shift-F11**.



- **Variables locales.**

Cuando estemos escribiendo código, la función de autocompletar el nombre de las variables con Ctr-Barra espaciadora también está disponible dentro de las funciones (para las variables locales).

Cuando estamos depurando, en la ventana `Variables Locales` podemos ver el valor de las variables locales de la función activa en ese momento. Obviamente, también aparecen los parámetros formales ya que son variables locales a la función. Comprobadlo ejecutando paso a paso la función `Max`.

Otra forma de ver el valor de una variable local durante la depuración es dejar el cursor del ratón encima de la variable y aparecerá una leyenda en amarillo con el valor que dicha variable tiene en ese momento (al igual que hacíamos con las variables del `main`)

- **Ir a definición.**

Otro recurso interesante del entorno de Visual Studio es el siguiente: situad el cursor sobre la llamada a una función, por ejemplo, sobre la llamada a `Max` desde el `main`. Haced click con el ratón derecha y seleccionad *Ir a definición* (o pulsad sobre el icono ). Automáticamente, el cursor se situará en la definición de la función. Para volver a la línea en la que estábamos, pulsaremos `Ctrl-` (navegación hacia atrás), o bien pincharemos sobre el icono .

- En la ventana *Vista de Clases*, si pinchamos sobre *Funciones y variables globales*, podremos ver las funciones definidas dentro del fichero `.cpp`. Haciendo doble click sobre cualquier función, el cursor se sitúa en el código de la función.

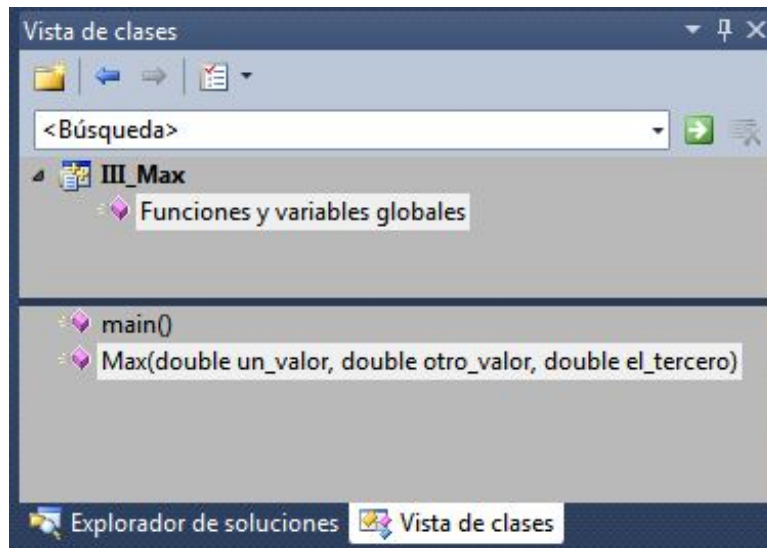


Figura 8: Vista de Clases

- Una vez hayamos definido una función, desde cualquier otra función definida con posterioridad, podemos invocarla escribiendo su nombre y la lista de parámetros actuales. Visual Studio permite el uso de `Ctrl-Barra espaciadora` para completar el nombre de la función llamada. Además, saldrá automáticamente una leyenda informativa con los nombres de los parámetros **formales**.

Apéndice: Saltar la depuración de ciertas funciones

Lo que sigue a continuación es para hacerlo en nuestra casa y no en esta sesión de prácticas.

Cuando se pulse F11 sobre una instrucción `cin`, `cout` o `system` el depurador de Visual Studio también entrará en el código fuente de dichos recursos. Para que no lo haga, debemos ejecutar el fichero `StepOver_VS_2010.reg` (se encuentra en el directorio `Otros Recursos Visual Studio` dentro del fichero `fp.zip` que se proporcionó al inicio de la asignatura). Para ello, basta hacer doble click sobre él desde el explorador del Sistema Operativo. Una vez hecho esto, volvemos a cargar Visual Studio. Esta operación sólo es necesario realizarla una vez en nuestro PC (en las aulas de prácticas no es necesario hacerlo). Si no funcionase la ejecución del fichero, hay que cambiar manualmente el registro de Windows, ejecutando `Regedit` y accediendo a la clave

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\VisualStudio\
10.0\NativeDE\StepOver]
```

y seleccionamos con el click derecho del ratón

Nuevo->Valor alfanumérico.

Añadimos sendas claves con los nombres `NoDepures1` y `NoDepures2` (por ejemplo) y con los siguientes valores:

```
std\:\:.*=NoStepInto
system=NoStepInto
```

Sesión 7

Funciones (II) - Clases (I)

► **Actividades a realizar en casa**

Actividad: Resolución de problemas.

Para esta sesión de prácticas continuaremos trabajando sobre modularización con funciones. Los ejercicios propuestos consisten en modularizar con funciones las soluciones a algunos ejercicios que propuestos anteriormente (e incluso resueltos) en la Relación de Problemas II (página **RP-II.1**). Se proponen dos ejercicios sencillos para practicar con clases.

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

Los enunciados de los ejercicios propuestos para esta sesión se encuentran en la Relación de Problemas III (página **RP-III.1**):

- *Obligatorios:*
 - 9 (Progresión geométrica con funciones).
 - 12 (Número perfecto con funciones).
 - 25 (Número aureo con funciones)
 - 13 (Clase *Terna* - Apartados a,b,c y d)
 - 14 (Clase *Depósito bancario*)
- *Opcionales:*
 - 10 (Número triangular con funciones).
 - 11 (Multiplicación rusa con funciones)
 - 26 (Número narcisista y número feliz con funciones)

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

El profesor irá llamando a los alumnos para que expliquen al resto de compañeros los ejercicios de esta sesión.

Los alumnos trabajarán y resolverán los ejercicios que proponga el profesor para esta sesión presencial.

.

Sesión 8

Clases (II)

► Actividades a realizar en casa

Actividad: Resolución de problemas.

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

- *Obligatorios:*
 - 13 (Clase *Terna* - Apartados e y f)
 - 15 (Progresión geométrica -con una clase-)
 - 29 (Gaussiana -con una clase-)
 - 16 (Pareja de enteros)
 - 17 (Pareja de enteros con un menú principal)
 - 18 (Pareja de enteros con el menú principal como clase)
- *Opcionales:*
 - 19 (Fabricante)
 - 20 y 34 (Nomina)
 - 22 (MiReal)
 - 27 (MiEntero)

Inspección de objetos

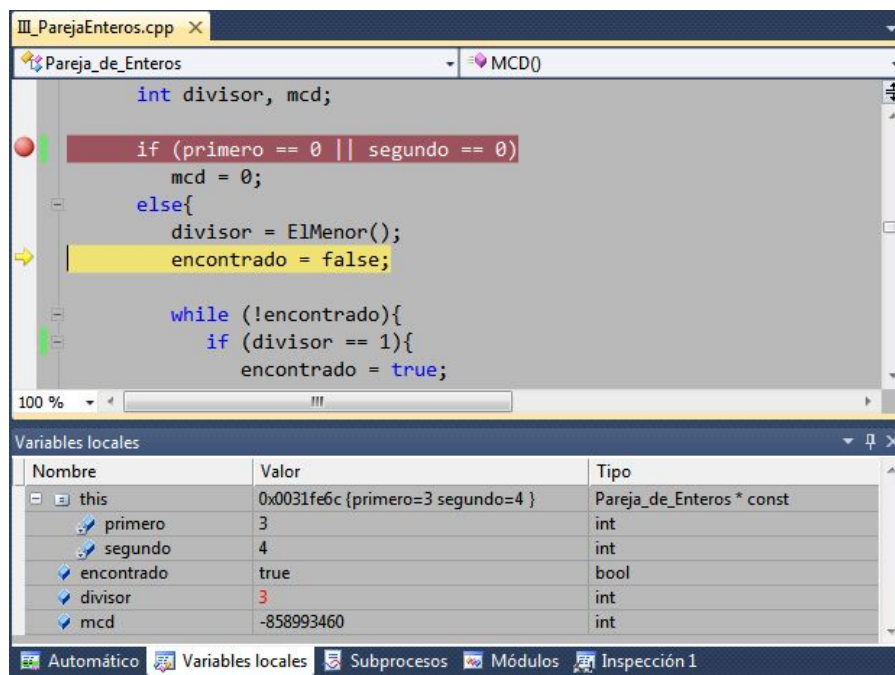
Cargar cualquier proyecto que declare, defina y use una clase. Incluir un objeto de la clase en la ventana de inspección, para ver cómo se pueden observar sus datos miembro desde el entorno de desarrollo. Seleccionad **Ver ->Vista de Clases** para visualizar las clases definidas en el proyecto. Haciendo doble click sobre un dato o método, nos situamos directamente en el código correspondiente de la clase.

Inspección de variable locales

Usar la solución del ejercicio 17. Incluid un punto de interrupción dentro del método MCD tal y como se indica en la figura. Ejecutar el programa introduciendo dos enteros cualesquiera y seleccionar la opción de calcular el máximo común divisor. Cuando la ejecución pare al llegar al punto de interrupción, abrir la ventana de inspección de variables locales:

Depurar -> Ventanas -> Variables Locales

Podremos ver el valor de las variables locales al método MCD, así como los datos miembro (desplegando el signo + que aparece junto al nombre `this`) De todas las variables que se muestran, la última en modificarse se mostrará en color rojo.



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

El profesor irá llamando a los alumnos para que expliquen al resto de compañeros los ejercicios de esta sesión.

Los alumnos trabajarán y resolverán los ejercicios que proponga el profesor para esta sesión presencial.

Sesión 9

Vectores (I)

► Actividades a realizar en casa

Actividad: Resolución de problemas.

Resuelve los siguientes ejercicios de la Relación de Problemas IV (página **RP-IV.1**).

Para la resolución de estos problemas recomendamos tomar como referencia la clase `MiVectorCaracteres` e ir ampliándola conforme se van resolviendo los ejercicios. De hecho, muchos de los ejercicios tienen como objetivo implementar métodos adicionales para esta clase, y en algunos ejercicios se emplean métodos desarrollados en otros ejercicios.

- **Obligatorios:**
 - 1 (Palíndromo)
 - 2 (Modifica componente)
 - 3 (Invierte)
 - 4 (Elimina componente)
 - 5 (Elimina las mayúsculas)
- **Opcionales:**
 - 6 (Elimina las mayúsculas eficientemente)

► Actividades a realizar en las aulas de ordenadores

El profesor irá llamando a los alumnos para evaluar los ejercicios de esta sesión.

Mientras tanto, los demás alumnos realizarán las siguientes tareas:

1. Crearán un proyecto nuevo llamado `Sesion9`.
2. Agregarán el fichero `PpalSesion9.cpp` y escribirán la clase `MiVectorCaracteres` ampliada con **todos** los métodos pedidos en los ejercicios de esta sesión
3. Escribirán la función `main()` de manera que ofrezca un **menú** que permita probar todos los métodos implementados.
4. Para gestionar el menú de opciones y la selección de la opción escribirán la clase `MenuSesion9`

.

Sesión 10

Vectores (II)

► **Actividades a realizar en casa**

Actividad: Resolución de problemas.

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

- *Obligatorios:*
 - 7 (Elimina repetidos)
 - 9 (Contar mayúsculas)
 - 10 (Contar mayúsculas - Clase Mayusculas)
- *Opcionales:*
 - 8 (Elimina exceso de blancos)

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

El profesor irá llamando a los alumnos para que expliquen al resto de compañeros los ejercicios de esta sesión.

Los alumnos trabajarán y resolverán los ejercicios que proponga el profesor para esta sesión presencial.

.

Sesión 11

Vectores (III)

► **Actividades a realizar en casa**

Actividad: Resolución de problemas.

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

- **Obligatorios:**
 - 9** (Cuenta mayúsculas) Resolver este problema con un vector de la STL en lugar de usar un objeto de la clase `MiVectorEnteros`
 - 10** (Contar mayúsculas - Clase `Mayusculas`) Resolver este problema con un vector de la STL en lugar de usar un vector local “clásico”
 - 11** (Come cocos)
 - 14** (Eratóstenes)
- **Opcionales:**
 - 12** (Permutación)
 - 13** (Fibonacci)

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

El profesor irá llamando a los alumnos para que expliquen al resto de compañeros los ejercicios de esta sesión.

Los alumnos trabajarán y resolverán los ejercicios que proponga el profesor para esta sesión presencial.

.

Sesión 12

Clases - Segunda parte (I)

► **Actividades a realizar en casa**

Actividad: Resolución de problemas.

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

- *Obligatorios:*
 - 1 (RLE)
 - 2 (Clases Circunferencia y Punto2D)
 - 3 (Generador de permutaciones aleatorias)
 - 4 (Permutación con dato miembro de tipo `MiVectorEnteros`)
 - 5 (Conjunto ordenado)
- *Opcionales:*
 - 6 (Subcamino débil)
 - 7 (Subcamino fuerte)

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

El profesor irá llamando a los alumnos para que expliquen al resto de compañeros los ejercicios de esta sesión.

Los alumnos trabajarán y resolverán los ejercicios que proponga el profesor para esta sesión presencial.

.

Sesión 13

Clases - Segunda parte (II)

► **Actividades a realizar en casa**

Actividad: Resolución de problemas.

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

- **Obligatorios:**

- 8 Colección de puntos y circunferencia.
- 14 (Métodos sobre una matriz rectangular de caracteres) y 15 (Matriz rectangular de caracteres simétrica). Estos dos ejercicios pueden resolverse en una sola entrega.
- 16 (Métodos sobre una matriz rectangular de enteros -Haced sólo los apartados **b** y **c-**) y 17 (Semejanza *matriz-vector*). Estos ejercicios pueden resolverse en una sola entrega.
- 18 (clase Palabra)
- 19 (clase Frase como vector de objetos de clase Palabra)

- **Opcionales:**

- 20 (Exámenes tipo test)

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

El profesor irá llamando a los alumnos para que expliquen al resto de compañeros los ejercicios de esta sesión.

Los alumnos trabajarán y resolverán los ejercicios que proponga el profesor para esta sesión presencial.

.

Sesión 14

Recursividad

► **Actividades a realizar en casa**

Actividad: Resolución de problemas.

Resolved los siguientes ejercicios de la relación de problemas VI:

- *Obligatorios:*
 - 1 (Sumar dígitos)
 - 2 (División entera)
 - 3 (Palíndromo)
 - 4 (k-ésimo)
 - 5 (Algoritmo de Euclides)
- *Opcionales:*
 - 6 (Contiene subcadena desde inicio)
 - 7 (Contiene subcadena)
 - 11 (Bisección)
 - 13 (Mayores pares)

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

El profesor irá llamando a los alumnos para que expliquen al resto de compañeros los ejercicios de esta sesión.

Los alumnos trabajarán y resolverán los ejercicios que proponga el profesor para esta sesión presencial.

.

FUNDAMENTOS DE PROGRAMACIÓN

Grado en Ingeniería Informática



Relaciones de Problemas.

Curso 2013/2014

RELACIÓN DE PROBLEMAS I. Introducción a C++

Problemas Básicos

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

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

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

Dificultad Baja.

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

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

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

A continuación, el programa debe imprimir en pantalla el valor de la variable `total`. Tanto el `capital` como el interés serán valores reales. Supondremos que el usuario introduce el interés como un valor real entre 0 y 100, es decir, un interés del 5,4 % se introducirá como 5.4. También supondremos que lo introduce correctamente, es decir, que sólo introducirá valores entre 0 y 100.

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

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

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

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

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

¿Por qué no funciona? Buscad una solución.

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

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

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

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

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

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

Cambiad ahora el valor de la constante PI por el de 3.1415927, re-compilad y ejecutad.

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

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

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

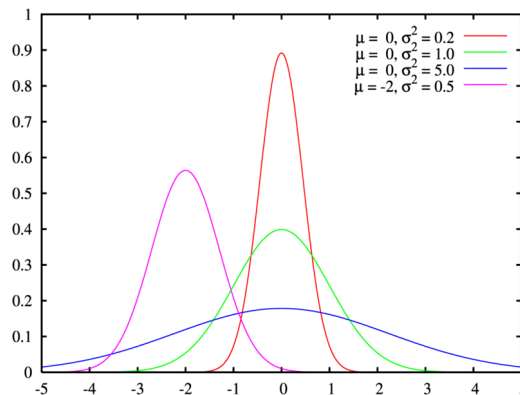
El parámetro μ se conoce como *esperanza* o *media* y σ como *desviación típica* (*mean* y *standard deviation* en inglés). Para definir la función matemática e usad la función `exp` de la biblioteca `cmath`. En la misma biblioteca está la función `sqrt` para calcular la raíz cuadrada. Para elevar un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

`pow(base, exponente)`

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

<http://www.danielsoper.com/statcalc/calc54.aspx>

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



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

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

Importante: No repetid cálculos ya realizados.

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

7. Redactar un algoritmo para calcular la media aritmética muestral y la desviación estándar (o típica) muestral de las alturas de tres personas. Éstos valores serán reales (de tipo `double`)

$$\bar{X} = \frac{1}{3} \sum_{i=1}^3 x_i, \quad \sigma = \sqrt{\frac{1}{3} \sum_{i=1}^3 (x_i - \bar{X})^2}$$

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

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

8. Realizar un programa que declare las variables `x`, `y` y `z`, les asigne los valores 10, 20 y 30 e intercambien entre sí sus valores de forma que el valor de `x` pasa a `y`, el de `y` pasa a `z` y el valor de `z` pasa a `x` (se pueden declarar variables auxiliares aunque se pide que se use el menor número posible).

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

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

y segundos dentro de su rango correspondiente. Por ejemplo, dadas 10 horas, 119 minutos y 280 segundos, debería dar como resultado 12 horas, 3 minutos y 40 segundos. En el caso de que nos salgan más de 24 horas, daremos también los días correspondientes (pero ya no pasamos a ver los meses, años, etc)

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

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

10. Realizad el ejercicio del reparto de la ganancia de un producto, pero cambiando el tipo de dato de la ganancia total a `int` (el resto de variables siguen siendo `double`)

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

11. Realizad el ejercicio del cálculo de la desviación típica, pero cambiando el tipo de dato de las variables x_i a `int`.

Nota: Para no tener problemas en la llamada a la función `pow` (en el caso de que se haya utilizado para implementar el cuadrado de las diferencias de los datos con la media), obligamos a que la base de la potencia sea un real multiplicando por 1.0, por lo que la llamada quedaría en la forma `pow(base*1.0, exponente)`

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

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

- a)

```
int resultado, entero1, entero2;
entero1 = 123456789;
entero2 = 123456780;
resultado = entero1 * entero2;
```
- b)

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

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

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

Nota. Si se desea ver el contenido de una variable real con `cout`, es necesario que antes de hacerlo, se establezca el número de decimales que se quieren mostrar en pantalla. Hacedlo escribiendo la sentencia `cout.precision(numero_digitos);`, en cualquier sitio del programa antes de la ejecución de `cout << real1 << "," << real2;`. Hay que destacar que al trabajar con reales siempre debemos asumir representaciones aproximadas por lo que no podemos pensar que el anterior valor `numero_digitos` esté indicando un número de decimales con representación exacta.

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

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

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

14. Supongamos el siguiente código:

```
int entero;
char caracter;

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

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

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

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

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

15. Razonar sobre la falsedad o no de las siguientes afirmaciones:

- a) 'c' es una expresión de caracteres.
- b) $4 < 3$ es una expresión numérica.
- c) $(4+3) < 5$ es una expresión numérica.

d) `cout << a;` da como salida la escritura en pantalla de una `a`.

e) ¿Qué realiza `cin >> cte`, siendo `cte` una constante entera?

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

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

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

Escribid una expresión lógica que nos informe cuando un año es bisiesto. Los años bisiestos son aquellos que o bien son divisibles por 4 pero no por 100, o bien son divisibles por 400.

Nota: Cuando se imprime por pantalla (con `cout`) una expresión lógica que es `true`, se imprime 1. Si es `false`, se imprime un 0. En el tema 2 veremos la razón.

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

17. Indique qué tipo de dato usaría para representar:

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

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

Problemas de Ampliación

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

Dificultad Baja.

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

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

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

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

Algunas funciones de `cmath`

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

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

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

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

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

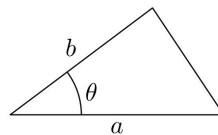
Dificultad Baja.

20. 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

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

Dificultad Baja.

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



Tened en cuenta que el argumento de la función `sin` va en radianes por lo que habrá que transformar los grados del ángulo en radianes.

Dificultad Baja.

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

3 5 1

Dificultad Media.

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

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

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

Dificultad Media.

RELACIÓN DE PROBLEMAS II. Estructuras de Control

Problemas Básicos

1. Ampliad el ejercicio 7 de la relación de problemas I, para que, una vez calculada la media y la desviación, el programa imprima por cada uno de los valores introducidos previamente, si está por encima o por debajo de la media. Por ejemplo:

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

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

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

2. Cread un programa que incluya una variable `edad` en la que guardamos la edad de una persona (como una variable entera) y otra variable `ingresos`, en la que almacenamos sus ingresos (como un real). Subid sus ingresos en un 5% si es un jubilado con unos ingresos inferiores a 300 euros e imprimid el resultado por pantalla. En caso contrario imprimid el mensaje "No es aplicable la subida". En ambos casos imprimid los ingresos resultantes.

Resolved este ejercicio de dos formas:

- a) En primer lugar, mezclando E/S (entradas y salidas) con cálculos dentro de la misma estructura condicional
- b) En segundo lugar, separándolos en bloques distintos

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

3. Realizar un programa en C++ que lea dos valores enteros desde teclado y diga si cualquiera de ellos divide o no (de forma entera) al otro. En este problema no hace falta decir quién divide a quién. Supondremos que los valores leídos desde teclado son ambos distintos de cero y se pueden mezclar cálculos con entradas y salidas.

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

4. Se quiere leer un carácter `letra_original` desde teclado, y comprobar con una estructura condicional si es una letra mayúscula. En dicho caso, hay que calcular la minúscula correspondiente en una variable llamada `letra_convertida`. En cualquier otro caso, le asignaremos a `letra_convertida` el valor que

tenga `letra_original`. Finalmente, imprimiremos en pantalla el valor de `letra_convertida`. No pueden usarse las funciones `tolower` ni `toupper` de la biblioteca `cctype`.

a) Se propone una primera solución como sigue:

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

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

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

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

b) Usamos en primer lugar un `string`:

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

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

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

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

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

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


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

if (es_mayuscula)
    letra_convertida = letra_original + DISTANCIA_MAY_MIN;
else
    letra_convertida = letra_original;

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

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

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

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

5. Queremos modificar el ejercicio 4 para leer un carácter `letra_original` desde teclado y hacer lo siguiente:

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

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

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

estaremos cometiendo el tipo de error indicado en el ejercicio 4, ya que si le asignamos un valor a una de las variables lógicas, no se lo asignamos a la otra y se queda con basura. Para resolverlo, planteamos la siguiente solución:

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

o si se prefiere, de una forma más concisa:

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

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

```
es_minuscula = !es_mayuscula;
```

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

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

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

Finalidad: Plantear una estructura condicional anidada. Dificultad Baja.

6. Para salir de una crisis económica, el gobierno decide:

- Bajar un 3 % la retención fiscal a los autónomos
- Para las rentas de trabajo:
 - Se sube un 3 % la retención fiscal a todos los pensionistas
 - Al resto de los trabajadores se le sube un 2 % la retención fiscal. Una vez hecha esta subida lineal del 2 %, se le aplica (sobre el resultado anterior) las siguientes subidas adicionales, dependiendo de su estado civil y niveles de ingresos:
 - Subir un 6 % la retención fiscal a las rentas de trabajo inferiores a 20.000 euros

RELACIÓN DE PROBLEMAS II. Estructuras de Control

- Subir un 8 % la retención fiscal a los casados con rentas de trabajo superiores a 20.000 euros
- Subir un 10 % la retención fiscal a los solteros con rentas de trabajo superiores a 20.000 euros

Cread un programa que trabaje con las variables `renta_bruta`, `renta_neta`, `retencion` y calcule la nueva retención a aplicar según los criterios anteriores, la aplique a la renta bruta para obtener la renta neta e imprima el resultado.

En este ejercicio no haremos ninguna lectura con `cin`, sino que declararemos las variables necesarias (por ejemplo `retencion_fiscal`, `es_autonomo`, etc) y les asignaremos los valores que se deseen directamente dentro del código.

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

Finalidad: Plantear una estructura condicional anidada. Dificultad Baja.

7. Una compañía aérea establece el precio del billete como sigue: en primer lugar se fija una tarifa base de 150 euros, la misma para todos los destinos. Si el destino está a menos de 200 kilómetros, el precio final es la tarifa inicial. Para destinos a más de 200 Km, se suman 10 céntimos por cada kilómetro de distancia al destino (a partir del Km 200).

En una campaña de promoción se va a realizar una rebaja lineal de 15 euros a todos los viajes. Además, se pretenden añadir otras rebajas y se barajan las siguientes alternativas de políticas de descuento:

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

Cread un programa para que lea el número de kilómetros al destino y si el billete corresponde a un cliente previo de la empresa. Calcular el precio final del billete con las siguientes políticas de descuento:

- Aplicando c) de forma adicional a los descuentos a) y b)
- Aplicando c) de forma exclusiva con los anteriores, es decir, que si se aplica c), no se aplicaría ni a) ni b)

Finalidad: Plantear una estructura condicional anidada. Dificultad Media.

8. En el ejercicio 5 se han usado dos variables lógicas (`es_mayuscula`, `es_minuscula`), que nos proporciona la distinción entre 4 casos distintos (VV, VF, FV, FF). Sin embargo, sólo queremos distinguir tres posibilidades, a saber, es mayúscula, es minúscula y no es un carácter alfabético. Para ello, volved a

RELACIÓN DE PROBLEMAS II. Estructuras de Control

resolver el ejercicio 5 substituyendo las dos variables lógicas por un tipo enumerado adecuado.

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

9. Realizar un programa que lea desde teclado un entero tope e imprima en pantalla todos sus divisores. Para obtener los divisores, basta recorrer todos los enteros menores que el valor introducido y comprobar si lo dividen. A continuación, mejorar el ejercicio obligando al usuario a introducir un entero positivo, usando un filtro con un bucle post test (do while).

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

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

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

11. Modifiquemos el ejercicio 2 del capital y los intereses de la primera relación. Supongamos ahora que se quiere reinvertir todo el dinero obtenido (el original C más los intereses producidos) en otro plazo fijo a un año. Y así, sucesivamente. Construid un programa para que lea el capital, el interés y un número de años N , y calcule e imprima todo el dinero obtenido durante cada uno de los N años, suponiendo que todo lo ganado (incluido el capital original C) se reinvierte a plazo fijo durante el siguiente año. El programa debe mostrar una salida del tipo:

```
Total en el año número 1 = 240
Total en el año número 2 = 288
Total en el año número 3 = 345.6
.....
```

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

12. Sobre el mismo ejercicio del capital y los intereses, construid un programa para calcular cuantos años han de pasar hasta llegar a doblar, como mínimo, el capital inicial. Los datos que han de leerse desde teclado son el capital inicial y el interés anual.

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

13. Se pide leer un carácter desde teclado, obligando al usuario a que sea una letra mayúscula. Para ello, habrá que usar una estructura repetitiva `do while`, de forma que si el usuario introduce un carácter que no sea una letra mayúscula, se le volverá a pedir otro carácter. Calculad la minúscula correspondiente e imprimidla en pantalla. No pueden usarse las funciones `tolower` ni `toupper` de la biblioteca `cctype`. Si se quiere, se puede usar como base el proyecto que resolvió el ejercicio 13 de la relación de problemas I.

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

14. Calcular mediante un programa en C++ la función potencia x^n , y la función factorial $n!$ con n un valor entero y x un valor real. No pueden usarse las funciones de la biblioteca `cmath`.

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

$$0! = 1$$

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

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

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

El combinatorio de n sobre m (con $n \geq m$) es un número entero que se define como sigue:

$$\binom{n}{m} = \frac{n!}{m! (n - m)!}$$

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

16. Un número entero de n dígitos se dice que es **narcisista** si se puede obtener como la suma de las potencias n -ésimas de cada uno de sus dígitos. Por ejemplo 153 y 8208 son números narcisistas porque $153 = 1^3 + 5^3 + 3^3$ y $8208 = 8^4 + 2^4 + 0^4 + 8^4$. Construir un programa que, dado un número entero positivo, nos indique si el número es o no narcisista. Si se va a usar la función `pow`, no pueden ser ambos argumentos enteros, por lo que forzaremos a que la base (por ejemplo) sea `double`, multiplicando por 1.0.

Finalidad: Ejercitar los bucles. Dificultad Media.

17. 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:

RELACIÓN DE PROBLEMAS II. Estructuras de Control

- a) Modificad la solución del ejercicio 9 de forma que el filtro de entrada usado para leer la variable `tope`, se haga con un bucle pre-test `while`.
- b) Modificad la solución del ejercicio 11 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.

18. Una empresa que tiene tres sucursales decide llevar la contabilidad de las ventas de sus productos a lo largo de una semana. Para ello registra cada venta con tres números, el identificador de la sucursal (1,2 ó 3), el código del producto (1, 2 ó 3) 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

```
1 2 10
1 2 4
1 1 1
1 1 1
1 3 2
2 2 15
2 2 6
2 1 20
3 3 40
-1
```

Se puede ver que la sucursal que más productos ha vendido es la número 2 con 41 unidades totales. La salida del programa deberá seguir exactamente el siguiente formato:

```
SUCURSAL: 2
VENTAS: 41
```

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.

19. Supongamos una serie numérica cuyo término general es:

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

RELACIÓN DE PROBLEMAS II. Estructuras de Control

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

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

Se proponen dos alternativas:

- Realizad la suma de la serie usando la función `pow` para el cómputo de cada término a_i . Los argumentos de `pow` no pueden ser ambos enteros, por lo que forzaremos a que la base (por ejemplo) sea `double`, multiplicando por `1.0`.
- Si analizamos la expresión algebraica de la serie numérica, nos damos cuenta que es una *progresión geométrica* ya que cada término de la serie queda definido por la siguiente expresión:

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

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

Cread el programa usando esta fórmula. NO puede utilizarse la función `pow`.

¿Qué solución es preferible en términos de eficiencia?

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

20. Reescribid la solución a los ejercicios 9, 11 y 19 usando un bucle `for`

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

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

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

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

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

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

RELACIÓN DE PROBLEMAS II. Estructuras de Control

blanco, por ejemplo). Realizar un programa que lea una secuencia de números naturales terminada con un número negativo y la codifique mediante el método RLE. Leed los datos desde un fichero externo, tal y como se explica en la página 34.

Entrada:	1 1 1 2 2 2 2 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.

23. Sobre la solución del ejercicio 11 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 de 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.

24. Cread un programa que ofrezca en pantalla la siguiente salida:

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

Finalidad: Ejercitar los bucles anidados. Dificultad Baja.

25. Cread un programa que ofrezca en pantalla la siguiente salida:

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

Finalidad: Ejercitar los bucles anidados. Dificultad Media.

26. Modificad los dos ejercicios anteriores para que los valores inicial y final (1 y 6 en el ejemplo) puedan ser cualesquiera introducidos desde el teclado.

Finalidad: Ejercitar los bucles anidados. Dificultad Media.

27. Se dice que un número natural es feliz si cumple que si sumamos los cuadrados de sus dígitos y seguimos el proceso con los resultados obtenidos, finalmente obtenemos uno (1) como resultado. Por ejemplo, el número 203 es un número feliz ya que $2^2 + 0^2 + 3^2 = 13 \rightarrow 1^2 + 3^2 = 10 \rightarrow 1^2 + 0^2 = 1$.

Se dice que un número es feliz de grado k si se ha podido demostrar que es feliz en un máximo de k iteraciones. Se entiende que una iteración se produce cada vez que se elevan al cuadrado los dígitos del valor actual y se suman. En el ejemplo anterior, 203 es un número feliz de grado 3 (además, es feliz de cualquier grado mayor o igual que 3)

Escribir un programa que diga si un número natural n es feliz para un grado k dado de antemano. Tanto n como k son valores introducidos por el usuario.

Finalidad: Ejercitar los bucles anidados. Dificultad Media.

Problemas de Ampliación

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

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

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

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



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

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

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

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

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

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

RELACIÓN DE PROBLEMAS II. Estructuras de Control

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

Dificultad Media.

29. Cread un programa que acepte el número de un año e indique si es bisiesto o no. Un año es bisiesto si es múltiplo de cuatro, pero no de cien. Excepción a la regla anterior son los múltiplos de cuatrocientos que siempre son bisiestos. Son bisiestos: 1600, 1996, 2000, 2004. No son bisiestos: 1700, 1800, 1900, 1998, 2002.

Dificultad Baja.

30. 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.

31. 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 $x = -50, -48, \dots, 48, 50$ $y = -40, -39, \dots, 39, 40$, 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.

32. 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.

33. Diseñar un programa que lea caracteres desde la entrada y los muestre en pantalla, hasta que se pulsa el '.' y diga cuantos separadores se han leído (espacios en blanco ' ', tabuladores '\t' y caracteres de nueva línea '\n').

Dificultad Baja.

34. 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 II. Estructuras de Control

35. 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.

36. 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.

37. Diseñar un programa para jugar a adivinar un número. El juego tiene que dar pistas de si el número introducido por el jugador está por encima o por debajo del número introducido. Como reglas de parada considerad a) que haya acertado o b) se haya hartado y decida terminar (escoged cómo se quiere que se especifique esta opción)

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

Dificultad Media.

38. 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 $6 = 1 + 2 + 3$. Una forma de obtener los números triangulares es a través de la fórmula $\frac{n(n+1)}{2} \quad \forall n \in \mathbb{N}$. Se pide construir un programa que obtenga todos los números triangulares que hay menores que un entero k introducido desde teclado, sin aplicar la fórmula anterior.

Dificultad Baja.

39. Escribir un programa que lea una secuencia de números enteros en el rango de 0 a 100 terminada en un número mayor que 100 o menor que 0 y encuentre la subsecuencia de números ordenada, de menor a mayor, de mayor longitud. El programa nos debe decir la posición donde comienza la subsecuencia y su longitud. Por ejemplo, ante la entrada siguiente:

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

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

Dificultad Media.

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

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

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

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

Dificultad Media.

41. Construid un programa para comprobar si las letras de una palabra se encuentran dentro de otro conjunto de palabras. Los datos se leen desde un fichero de esta manera: 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 el 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.
- Si encontramos una letra en una palabra, ya no buscaremos más letras en ella.

RELACIÓN DE PROBLEMAS II. Estructuras de Control

- 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.

42. 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.

43. 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.

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

$\phi = \frac{1 + \sqrt{5}}{2}$. Se pueden construir aproximaciones al número áureo mediante la fórmula $a_n = \frac{fib(n+1)}{fib(n)}$ siendo $fib(n)$ el término n -ésimo de la sucesión de fibonacci que se define como:

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

Escribir un programa que calcule el menor valor de n que hace que la aproximación dada por a_n difiera en menos de δ del número ϕ , sabiendo que $n \geq 1$. La entrada del programa será el valor de δ y la salida el valor de n . Por ejemplo, para un valor de $\delta = 0,1$ el valor de salida es $n = 3$

Dificultad Media.

RELACIÓN DE PROBLEMAS III. Funciones y Clases

Problemas Básicos

Problemas sobre funciones

1. Encuentre los errores de las siguientes funciones:

```
int Suma5 (int entero) {           void Imprime(double valor) {
    if (0 == entero)                double valor;
        return 0;
    else                            cout << valor;
        entero = entero + 5;        }
}

void Cuadrado (int entero) {        bool EsPositivo(int valor) {
    return entero*entero;            if (valor > 0)
}                                    return true;
                                    }
}
```

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

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

```
double Max(double primero, double segundo, double tercero)
```

Construid un programa principal que llame a dicha función con los parámetros actuales que se quiera.

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

3. Reescribir el programa construido para implementar la solución al ejercicio 14 (factorial y potencia) de la Relación de Problemas II modularizando con funciones.

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

4. Implementar la función que calcula el número combinatorio (ejercicio 15 de la Relación de Problemas II).

Finalidad: Entender el ámbito de los parámetros de las funciones, así como las llamadas entre funciones. Dificultad Baja.

5. Se pide definir las siguientes funciones y crear un programa principal que las llame:

- a) `EsMinuscula` comprueba si un carácter pasado como parámetro es una minúscula. A dicho parámetro, llámalo `una_letra`.
- b) `Convierte_a_Mayuscula` comprueba si un carácter pasado como parámetro es minúscula (para ello, debe llamar a la función anterior), en cuyo caso lo transforma a mayúscula. Si el carácter no es minúscula debe dejar la letra igual. A dicho parámetro, llámalo `caracter`.

Esta función hace lo mismo que la función `tolower` de la biblioteca `cctype`

Observad que el parámetro `una_letra` de la función `EsMinuscula` podría llamarse igual que el parámetro `caracter` de la otra función. Esto es porque están en ámbitos distintos y para el compilador son dos variables distintas. Haced el cambio y comprobarlo.

Finalidad: Entender el ámbito de los parámetros de las funciones, así como las llamadas entre funciones. Dificultad Baja.

6. Vamos a añadir una función al ejercicio 5 de esta relación de problemas. Se pide implementar la función `Transforma`, 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 minúscula, simplemente devuelve el mismo carácter que recibió.

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?

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

7. En el ejercicio 5 de la Relación de Problemas I (página RP-I.1) se vio cómo obtener el valor de ordenada asignado por la función gaussiana, sabiendo el valor de abscisa x . Recordemos que esta función matemática dependía de dos parámetros μ (esperanza) y σ (desviación), y venía definida por:

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

Cread una función para calcular el valor de la función gaussiana en un punto arbitrario leído desde teclado. Escribid un programa que haga uso de esa función.

Hacedlo de dos maneras: a) con una función que admita un sólo parámetro (x) y con una función que admita tres parámetros (x , μ y σ).

Finalidad: Entender el ámbito de los parámetros de las funciones, el uso de constantes globales o locales, el uso de funciones completamente parametrizadas, así como las llamadas entre funciones. Dificultad Media.

8. Implementar una solución más general que la pedida en el ejercicio 21 de la Relación de Problemas II (página RP-II.1) para el cálculo de la sumatoria de los elementos de una serie. Ahora queremos que se calculen **independientemente** el elemento i -ésimo de la serie y la sumatoria de los primeros k elementos (k es un valor que introduce el usuario) de la serie.

Implementar de la manera más eficiente posible el cálculo del elemento i -ésimo.

Finalidad: Entender el ámbito de los parámetros de las funciones, así como las llamadas entre funciones. Dificultad Baja.

9. Cread las siguientes funciones relacionadas con la progresión geométrica que se vio en el ejercicio 19 de la Relación de Problemas II (página RP-II.1). Analizad cuáles deben ser los parámetros a estas funciones.

- a) Una función `SumaHasta` que calcule la suma de los primeros k valores de una progresión geométrica.

Cread un programa principal que llame a dicha función.

- b) Cambiemos ahora la implementación de la anterior función `SumaHasta`. En vez de usar un bucle aplicamos la siguiente fórmula, que calcula la sumatoria aplicando únicamente cinco operaciones:

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

Es muy importante remarcar que `main` no cambia: hemos cambiado la implementación de la función y lo hemos podido hacer sin necesidad de cambiar la función

`main`, ya que ésta no tenía acceso al código que hay dentro de la función. Esto es *ocultación de información* (recordar las clases de teoría).

Nota. La potenciación (en nuestro ejemplo r^k) 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 de la función `main`.

- c) Añadid una función llamada `SumaHastaInfinito` para que calcule la suma hasta infinito, según la fórmula:

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

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

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

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

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

10. Reescribir el programa construido para implementar la solución al ejercicio 38 (Triangular) de la Relación de Problemas II modularizando con funciones.

Finalidad: Entender el ámbito de los parámetros de las funciones, así como las llamadas entre funciones. Dificultad Baja.

11. Reescribir el programa construido para implementar la solución al ejercicio 40 (Multiplicación rusa) de la Relación de Problemas II modularizando con funciones.

Finalidad: Entender el ámbito de los parámetros de las funciones, así como las llamadas entre funciones. Dificultad Baja.

12. Reescribir el programa construido para implementar la solución al ejercicio 42 (Perfecto) de la Relación de Problemas II modularizando con funciones.

Finalidad: Entender el ámbito de los parámetros de las funciones, así como las llamadas entre funciones. Dificultad Baja.

Problemas sobre clases

13. Se pide crear una clase *Terna* que represente un conjunto de 3 valores de tipo *double*, sobre los que se quiere realizar las operaciones de calcular el máximo, el mínimo y la media aritmética. Para ello, haced lo siguiente:

- a) Definid en primer lugar la clase con tres datos miembro públicos llamados *uno*, *dos* y *tres*. A esta primera versión de la clase, dadle el nombre

Terna_vs1

Añadid un método público *Max*. Este método tendrá la siguiente cabecera:

```
double Max()
```

Observad que el método *Max* actúa sobre los datos de la clase (*uno*, *dos* y *tres*) por lo que no tiene parámetros. Observad la diferencia con la cabecera del ejercicio 2 en el que *Max* era una función que estaba dentro de ninguna clase.

- b) Añadid un método para calcular el mínimo de los tres valores.
c) Añadid ahora un método para calcular la media aritmética de los tres valores.
d) Cread un *main* que llame a los métodos anteriores.

Comprobad que podemos modificar directamente los datos miembro públicos desde el *main*, lo cual no es muy recomendable como ya se ha visto en clase de teoría. Para resolver este problema, hacemos lo siguiente:

- e) Copiad el código de la clase anterior y copiadlo en una nueva clase llamada

Terna_vs2

En esta nueva versión, declarad los datos miembro como *private*, en vez de *public*. Comprobad que ya no podemos acceder a ellos desde el *main*, por lo que necesitamos asignarles un valor de otra forma. Para ello, haced lo siguiente:

- i) Añadid un constructor a la clase *Terna_vs2*. En dicho constructor, pasaremos los tres valores que habrá que asignárselos a los datos miembro.
Cread el objeto correspondiente desde el *main* (el compilador nos obligará a pasar los tres valores en el constructor)
ii) Añadid tres métodos (*GetPrimero()*, *GetSegundo()*, *GetTercero()*) que devuelvan los correspondientes datos miembro, para que así pueda consultarse su valor desde el *main*.
f) Supongamos que creamos un objeto con los valores 2, 5, 7. Si ahora quisiéramos cambiar de conjunto de datos y trabajar con 3, 9, 1, por ejemplo, tendríamos que crear otro objeto distinto. Esto es completamente lícito. En cualquier caso, dependiendo del problema que estemos abordando, podríamos estar interesados en permitir cambiar los datos del objeto (su *estado*) y así no tener que

crear otro objeto distinto. Para ello, debemos proporcionar dicha funcionalidad a la clase. Lo hacemos de dos formas alternativas:

- Una alternativa: creamos la clase

Terna_vs3

que contendrá el mismo código que `Terna_vs2` y le añadimos un método `SetValores` que cambie los tres valores a la vez (como ha cambiado el nombre de la clase, habrá que cambiar el nombre del constructor). Cread un objeto de esta clase desde el `main` y llamad a sus métodos.

En esta solución, asignamos (dentro del constructor) un primer valor a cada uno de los tres datos miembro y posteriormente permitimos su modificación a través del método `SetValores`. Sólo permitimos la modificación conjunta de los tres datos miembro a la misma vez.

- Otra alternativa: creamos la clase

Terna_vs4

para acceder a cada uno de los valores de forma independiente. Para ello, le añadimos tres métodos nuevos `SetPrimero`, `SetSegundo` y `SetTercero`, y el método `SetValores` lo declaramos `private` y lo llamamos dentro del constructor.

Cread un objeto de esta clase desde el `main` y llamad a sus métodos.

En esta solución, los tres datos miembro sólo los podemos asignar a través de un método específico para cada uno de ellos.

Observad la flexibilidad que me ofrece el concepto de clase. El programador decide cómo quiere que se acceda a los datos miembro: prohibiendo cambiar los datos miembro una vez que se ha construido el objeto (versión 2) o permitiéndolo, bien todos a la vez (versión 3) o uno a uno (versión 4).

Finalidad: Trabajar con los conceptos básicos de clases. Dificultad Baja.

A partir de ahora, todos los ejercicios deben resolverse utilizando únicamente datos miembro privados.



14. Se quiere construir una clase `DepositoSimulacion` para simular préstamos, ofreciendo la funcionalidad descrita en los ejercicios 11 (reinvierte capital e interés un número de años) y 12 (reinvierte capital e interés hasta obtener el doble de la cantidad inicial) de la Relación de Problemas II (página RP-II.6). Por tanto, la clase debe proporcionar, para un capital y unos intereses dados, métodos para:

- calcular el capital que se obtendrá al cabo de un número de años,
- 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:

- a) Si queremos modificar el capital y el interés una vez creado el objeto,
- b) Si queremos poder modificarlos de forma independiente,
- c) Si hay alguna restricción a la hora de asignar un valor al capital e interés,
- d) Si es mejor un método para calcular el número de años hasta obtener el doble de la cantidad inicial, o por el contrario es mejor un método para calcular el número de años hasta obtener una cantidad específica.

Finalidad: Diseñar la interfaz de una clase. Dificultad Baja.

15. En el ejercicio 9 de esta relación de problemas se definieron varias funciones para operar sobre una progresión geométrica. Definid ahora una clase para representar una progresión geométrica.
- a) Diseñad la clase pensando cuáles serían los datos miembro esenciales que definen una progresión geométrica, así como el constructor de la clase.
 - b) Definid los métodos `SumaHastaInfinito`, `SumaHasta` y `MultiplicaHasta`.
 - c) Cread un programa principal que llame a los anteriores métodos.

Finalidad: Comparar la ventaja de un diseño con clases a uno con funciones. Dificultad Baja.

16. Cread una clase llamada `ParejaEnteros` que represente una pareja de enteros cualesquiera. Añadid un constructor y los métodos que se estimen oportunos para asignarles valores a los enteros y para obtener el valor que éstos tengan. Las operaciones que queremos realizar sobre una pareja son calcular el mayor, el menor, comprobar si se dividen, ver quien es el dividendo y quien el divisor, intercambiarlos y calcular el máximo común divisor. Cread un `main` que llame a dichos métodos, para comprobar que están bien implementados.

¿Le añadiría a la clase `ParejaEnteros` un método para calcular el factorial de un número? ¿Por qué?

Finalidad: Diseñar la interfaz de una clase. Dificultad Baja.

17. Sobre el ejercicio 16 de esta relación de problemas se pide crear un programa principal más completo de la siguiente forma. En el `main` se leerán dos enteros y a continuación se ofrecerá al usuario un menú con las opciones para realizar cualquiera de las operaciones definidas en la clase `ParejaEnteros`. También se ofrecerá la posibilidad de imprimir el valor de cada uno de los dos enteros. Una vez ejecutada la operación, se volverá a mostrar el mismo menú, hasta que el usuario decida salir (pulsando la tecla 'X', por ejemplo).

Finalidad: Organizar un menú de opciones que se va repitiendo mientras el usuario no decida salir. Dificultad Media.

RELACIÓN DE PROBLEMAS III. Funciones y Clases

18. Transformad la solución del ejercicio 17 de esta relación de problemas para que la responsabilidad de la lectura de la opción introducida por el usuario recaiga en una clase a la que llamaremos `Menu`.

Finalidad: Ver una clase específica que gestione la entrada de datos. Dificultad Media.

19. Se quiere construir una clase para realizar la funcionalidad descrita en el ejercicio 10 de la Relación de Problemas I sobre la nómina del fabricante y diseñador (página RP-I.4). La clase debe proporcionar, al menos, los métodos `SalarioNetoDisenador` y `SalarioNetoFabricante`. El criterio del cálculo del salario es el mismo (el diseñador cobra el doble de cada fabricante) pero además, ahora se le va a aplicar una retención fiscal para el cómputo de los salarios netos. Dicha retención puede ser distinta para el fabricante y el diseñador.

Finalidad: Trabajar con datos miembro constantes. Dificultad Baja.

20. 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.
- 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.

21. Cread una clase para representar fracciones (ver transparencias de teoría). La clase debe proporcionar un método para normalizar la fracción, es decir, encontrar la fracción irreducible que sea equivalente. Para ello, tendrá que dividir el numerador y denominador por su máximo común divisor. El cálculo del MCD se hará de dos formas alternativas:

- Declarando MCD como una función global.
- Declarando MCD como un método privado.

En este caso, ambas alternativas son correctas. Permitiremos el uso de funciones globales cuando éstas sean muy genéricas, reutilizables en problemas muy dispares y que trabajen con datos básicos.

Finalidad: Diseñar la interfaz de una clase. Dificultad Media.

22. En C++, los tipos de datos simples como `int`, `char`, `double`, etc, no son clases. Por tanto, si declaramos una variable real en la forma `double real`, para calcular el seno de `real`, llamamos a una **función** en la forma

```
double real, resultado;  
real = 6.5;  
resultado = sin(real);
```

Estamos interesados en construir una clase llamada `MiReal` para trabajar con reales al *estilo de las clases*. Por tanto, para calcular el seno de `6.5`, deberíamos hacer algo como lo siguiente:

```
MiReal real(6.5);  
double resultado;  
resultado = real.Seno();
```

En esta caso, `Seno` es un **método** de la clase `MiReal` que devuelve un tipo `double`. Construid dicha clase, proporcionando los siguientes métodos:

- a) `Valor`. Este método debe devolver el valor del real (`6.5` en el ejemplo anterior)
- b) `Seno`. Para definirlo, llamad a la propia función `sin` de la biblioteca `cmath`. Debe devolver un `double`
- c) `ValorAbsoluto`. Devolverá un tipo `double` que contendrá el valor absoluto del real. No pueden utilizarse las funciones de `cmath`.

Cread un programa principal que llame a dichos métodos.

Observad que sería interesante que nuestro código fuese el siguiente:

```
MiReal real(6.5);  
MiReal resultado;  
resultado = real.Seno();  
resultado = resultado + real;
```

Es decir, que el método `Seno` devolviese un tipo `MiReal` en vez de un tipo `double`. Aunque esto es posible en C++, por ahora no disponemos de las herramientas necesarias para hacerlo. Por ejemplo, para hacer la asignación entre variables de tipo `MiReal` se necesita el concepto de *sobrecarga del operador de asignación* y para

RELACIÓN DE PROBLEMAS III. Funciones y Clases

realizar la suma de dos variables de tipo `MiReal` se necesita *sobrecargar* el operador `+` (se verá en el segundo cuatrimestre).

Nota. Este tipo de clases (`MiEntero`, `MiReal`, etc) son proporcionadas por Java de forma standard. En otros lenguajes como SmallTalk o plataformas como .NET (con lenguajes como Visual Basic, C#) no existe la diferencia entre tipos *básicos* (como `int`, `double`, etc) y todo son clases, siguiendo una filosofía orientada a objetos *pura*.

Finalidad: Entender las clases que actúan como **envolventes (wrappers)** de los tipos básicos.

Dificultad Baja.

Problemas de Ampliación

23. En el ejercicio 7 de la Relación de Problemas II (sobre los descuentos a aplicar a una tarifa aérea), se repetía en varias ocasiones el cómputo de un porcentaje, como por ejemplo:

```
tarifa_con_km_adicionales * DESCUENTO_CLIENTE_PREVIO/100.0;
```

Construid la función apropiada que realice la tarea del cómputo del porcentaje y reescribid la solución llamando a dicha función en todos los sitios donde sea necesario.

Añada las funciones que estime oportuno para evitar repetir código.

24. En los ejercicios 5 de la Relación de Problemas I (página RP-I.1) y 7 de esta misma Relación de Problemas (página RP-III.1) se trabajó sobre la función gaussiana.

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

$$\int_{-\infty}^x \text{gaussiana}(x) dx$$

tal y como puede comprobarse ejecutando el applet disponible en:

<http://dostat.stat.sc.edu/prototype/calculators/index.php3?dist=Normal>

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

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

dónde:

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

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

Cread una función para calcular el área hasta un punto cualquiera x , es decir, $CDF(x)$, usando la anterior aproximación.

Cread un programa principal que llame a las anteriores funciones.

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

25. Reescribir el programa construido para implementar la solución al ejercicio 44 (Número aureo) de la Relación de Problemas II modularizando con funciones.

Finalidad: Entender el ámbito de los parámetros de las funciones, así como las llamadas entre funciones.

Dificultad Media.

26. Reescribir los programas construidos para implementar las soluciones a los siguientes ejercicios de la Relación de Problemas II, modularizando con funciones las nuevas soluciones: 16 (narcisista) y 27 (feliz)

Dificultad Baja.

27. Construid la clase `MiEntero` que será utilizada en la forma siguiente:

```
MiEntero entero(6);  
int entero;  
resultado = entero.Factorial();
```

Debe proporcionar los siguientes métodos:

- a) `Valor`. Este método debe devolver el valor del entero (un tipo `int`).
- b) `Factorial`. Usad la solución del ejercicio 3 de esta misma Relación de Problemas.
- c) `EsNarcisista`. Usad la solución del ejercicio 26 de esta misma Relación de Problemas.
- d) `EsFeliz`. Usad la solución del ejercicio 26 de esta misma Relación de Problemas.

Cread un programa principal que llame a dichos métodos.

28. Construid la clase `MiCaracter` que será utilizada en la forma siguiente:

```
MiCaracter caracter('a');  
char resultado;  
resultado = entero.GetMayuscula();
```

Debe proporcionar los siguientes métodos:

- a) `Valor`. Este método debe devolver el valor del carácter.
- b) `EsMinuscula`. Devuelve un `bool`.
- c) `EsMayuscula`. Devuelve un `bool`.
- d) `EsLetra`. Devuelve un `bool`.

- e) `GetMayuscula`. Devuelve un `char` correspondiente al carácter convertido a mayúscula. Si el carácter no es minúscula, devuelve el mismo valor del `char`.
- f) `GetMinuscula`. El recíproco del anterior.

Cread un programa principal que llame a dichos métodos.

29. Recuperad el ejercicio 7 de esta relación de problemas sobre la función gaussiana. En vez de trabajar con funciones, plantead la solución con una clase.

Dificultad Media.

30. 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`.

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.

31. Se quiere construir una clase para representar la tracción de una bicicleta, es decir, el conjunto de estrella, cadena y piñón. Supondremos que el plato delantero tiene tres estrellas y el trasero tiene 7 piñones. 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 salto de uno o de dos.

Una vez hecha la clase y probada con un programa principal, modificadla 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:

- Primera estrella y piñón mayor o igual que 5
- Segunda estrella y piñón o bien igual a 1 o bien igual a 7
- Tercera estrella y piñón menor o igual que 4

Finalidad: Diseñar la interfaz de una clase. Trabajar con datos miembro constantes.. Dificultad Media.

32. Construid una clase llamada `MedidaAngulo` que represente una medida de un ángulo. Al igual que se hizo en el ejercicio 30, 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.

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

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

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

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

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

RELACIÓN DE PROBLEMAS IV. Vectores

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

```
class MiVectorCaracteres {

private:
    static const int TAMANIO = 50;
    char vector_privado[TAMANIO];
    int total_utilizados;

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

Importante:

- Para todos los ejercicios, se ha de diseñar una batería de pruebas.
- Para poder leer un espacio en blanco **no** puede emplear `cin >> caracter`, sino `caracter = cin.get()`. Cada vez que se ejecute `cin.get()` el compilador lee un carácter (incluido el espacio en blanco) desde la entrada de datos por defecto.

Problemas Básicos

1. Añadir un método con la cabecera:

```
bool EsPalindromo (void)
```

que nos diga 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.

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

2. Añadir un método con la cabecera:

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

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

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

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

3. Añadir el método `IntercambiaComponentes()` para intercambiar dos componentes del vector. Por ejemplo, si el vector contiene {'h', 'o', 'l', 'a'}, después de intercambiar las componentes 1 y 3, se quedaría con {'h', 'a', 'l', 'o'}.

¿Qué debería hacer este método si los índices no son correctos?

Añadir otro método llamado `Invierte()` para invertir el vector, de forma que si el vector contenía, por ejemplo, los caracteres {'m', 'u', 'n', 'd', 'o'}, después de llamar al método se quedará con {'o', 'd', 'n', 'u', 'm'}.

Nota: Para implementar este método, llamad a `IntercambiaComponentes()`.

Imprimir las componentes del vector desde `main()`, antes y después de llamar a dicho método. Observad que se repite el mismo tipo de código cuando se imprimen las componentes del vector. Ya lo arreglaremos en el tema siguiente.

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

4. Añadir el método `Elimina()` para eliminar el carácter que se encuentre en una determinada posición, de forma que si el vector contenía, por ejemplo, los caracteres {'h', 'o', 'l', 'a'}, después de eliminar la componente con índice 2 (la tercera) se quedará con {'h', 'o', 'a'}.

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

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

5. Añadir el método `EliminaMayusculas()` para eliminar todas las mayúsculas. Por ejemplo, después de aplicar dicho método al vector `{'S','o','Y',' ','y','0'}`, éste debe quedarse con `{'o',' ','y'}`.

En este ejercicio proponemos implementar el siguiente algoritmo:

```
Recorrer todas las componentes del vector
  Si la componente es una mayúscula, borrarla
```

Para borrar la mayúscula, se *desplazan* una posición a la izquierda todas las componentes que hay a su derecha. Para ello, llamad al método `Elimina()` que se ha definido en el ejercicio 4 de esta relación de problemas.

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

6. El algoritmo del ejercicio 5 es muy ineficiente ya que requiere trasladar muchas veces muchas posiciones. Proponer un algoritmo para resolver eficientemente este problema e implementarlo.

Consejo: Una forma de hacerlo es utilizar dos variables, `posicion_lectura` y `posicion_escritura` que nos vayan indicando, en cada momento, la componente que se está leyendo y el sitio dónde tiene que escribirse.

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

7. Añadir 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 el vector contiene `{'b','a','a','h','a','a','a','a','c','a','a','a','g'}` después de quitar los repetidos, se quedaría como sigue: `{'b','a','h','c','g'}`

Implementad los siguientes algoritmos para resolver este problema:

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

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

- b) El problema del algoritmo anterior es que usa un vector local, lo que podría suponer una carga importante de memoria si trabajásemos con vectores grandes. Por lo tanto, vamos a resolver el problema sin usar vectores locales.

Si una componente está repetida, **se borrará del vector**. Para borrar una componente, podríamos desplazar una posición a la izquierda, todas las componentes que estén a su derecha. El algoritmo quedaría:

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

- c) El anterior algoritmo nos obliga a desplazar muchas componentes cada vez que encontremos una repetida. Proponed una alternativa (sin usar vectores locales) para que el número de desplazamientos sea el menor posible.

Dificultad Media.

8. Añadir 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 el vector original es (' ', 'a', 'h', ' ', ' ', ' ', 'c'), que contiene una secuencia de tres espacios consecutivos, el vector resultante debe ser (' ', 'a', 'h', ' ', 'c').

Nota: Debe hacerse lo más eficiente posible.

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

Problemas de Ampliación

9. Realizar un programa que vaya leyendo caracteres hasta que se encuentre un punto '.'. Queremos contar el número de veces que aparece cada una de las letras mayúsculas.

Una posibilidad sería leer el carácter, y después de comprobar si es una mayúscula, ejecutar un conjunto de sentencias del tipo:

```
if (letra == 'A')
    contador_mayusculas[0] = contador_mayusculas[0] + 1;
else if (letra == 'B')
    contador_mayusculas[1] = contador_mayusculas[1] + 1;
else if (letra == 'C')
    contador_mayusculas[2] = contador_mayusculas[2] + 1;
else ....
```


RELACIÓN DE PROBLEMAS IV. Vectores

Sin embargo, este código es muy redundante. Proponed una solución e implementarla. Para resolver este ejercicio puede usarse o bien un vector clásico *-array-* de enteros, o bien un objeto de la clase `MiVectorEnteros`. La idea es que todos los `if-else` anteriores los podamos resumir en una única sentencia del tipo:

```
contador_mayusculas[indice] = contador_mayusculas[indice]+1;
```

si empleamos un vector clásico *-array-* o bien

```
contador_mayusculas.Modifica(indice,  
                              contador_mayusculas.Elemento(indice)+1);
```

si empleamos un objeto de la clase `MiVectorEnteros`.

Finalidad: Diseñar un vector en el que los índices de las componentes representan algo. Dificultad Media.

10. Sobre el ejercicio 9, cread una clase específica `Mayusculas` que implemente los métodos necesarios para realizar la cuenta de las mayúsculas. Lo que se pretende es que la clase proporcione los métodos siguientes:

```
void IncrementaCuenta (char mayuscula)  
int  CuantasHay (char mayuscula)
```

Dificultad Media.

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

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

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

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

Resolved este problema usando las siguientes aproximaciones:

- a) Definiendo una cadena local `a_buscar` con la cadena formada por la secuencia de movimientos a buscar y llamando al método `find()` de la clase `string`.
- b) Sin usar `find()`, accediendo directamente a las componentes de la cadena.

Dificultad Baja.

12. Cread una clase `Permutacion` para representar una permutación de enteros.

- Para almacenar los valores enteros usaremos como dato miembro privado un vector de la STL.
- Pasaremos como parámetro al constructor un vector de la STL con los valores enteros. Supondremos como precondition que el vector es una permutación correcta, es decir, que contiene todos los enteros sin repetir entre el mínimo y el máximo de dichos valores. Por ejemplo, $(2, 3, 6, 5, 4)$ es una permutación correcta pero no lo es $(7, 7, 6, 5)$ (tiene el 7 como valor repetido) ni tampoco $(7, 6, 4)$ (le falta el 5).
- Añadid un método que calcule el número de lecturas de dicha permutación. Una permutación de un conjunto de enteros, $C = \{1, \dots, n\}$, tiene k lecturas, si para leer sus elementos en orden creciente (de izquierda a derecha) tenemos que recorrer la permutación k veces. Por ejemplo, la siguiente permutación del conjunto $\{0, \dots, 8\}$:

4 0 8 1 2 5 3 6 7

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

Dificultad Media.

13. Escribir 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 . Para almacenar los enteros, se usará un dato miembro de tipo vector de la STL.

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, ...

programa debe definir una clase llamada `Fibonacci`. Al constructor se le pasará como parámetro el valor de n y la clase tendrá métodos para observar el valor de n y tres métodos:

- `void CalculaPrimeros(int tope)` para que calcule los tope primeros elementos de la sucesión.
- `int TotalCalculados()` que devuelva cuántos elementos hay actualmente almacenados (el valor tope del método anterior)
- `int Elemento(int indice)` para que devuelva el elemento índice-ésimo de la sucesión.

El programa principal quedaría de la forma:

```
int k = 100; int n = 4; int tope;

Fibonacci fibonacci(n);

fibonacci.CalculaPrimeros(k);
tope = fibonacci.TotalCalculados();    // tope = k

for (int i=0; i<tope; i++)
    cout << fibonacci.Elemento(i) << " ";
```

Dificultad Media.

14. ([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 “manual” consiste en escribir todos los números naturales comprendidos entre 2 y n y *tachar* los números que *no* son primos de la siguiente manera: el primero (el 2) se declara primo y se tachan todos sus múltiplos; se busca el siguiente número entero que no ha sido tachado, se declara primo y se procede a tachar todos sus múltiplos, y así sucesivamente. El proceso para cuando el cuadrado del número entero es mayor o igual que el valor de n .

El programa debe definir una clase llamada Eratostenes que tendrá tres métodos:

- void CalculaHasta(int n) para que calcule los primos menores que n .
- int TotalCalculados() que devuelva cuántos primos hay actualmente almacenados.
- int Elemento(int indice) para que devuelva el índice-ésimo primo.

El programa principal quedaría de la forma:

```
Eratostenes primos;
int n = 100; int tope;

primos.CalculaHasta(n);
tope = primos.TotalCalculados();

for (int i=0; i<tope; i++)
    cout << primos.Elemento(i) << " ";
```

Para almacenar los primos, se usará un dato miembro de tipo vector de la STL.

Dificultad Media.

15. Cread una clase `Frase` para almacenar un conjunto de caracteres. Internamente, como dato miembro privado, se usará una variable cadena de tipo `string` que almacenará los caracteres.

Añadid los siguientes métodos:

- `void EliminaBlancosIniciales()` para borrar todos los blancos iniciales.
- `void EliminaBlancosFinales()` para borrar todos los blancos finales.
- `int NumeroPalabras()` que indique cuántas palabras hay en la frase (una palabra está separada por otra por uno o más espacios en blanco)
- `void BorraPalabra(int k_esima)` para que borre la palabra k -ésima.
- `string Palabra(int k_esima)` que devuelva la k -ésima palabra de la cadena. Si hay menos de k palabras, devolverá la cadena vacía `" "`.

Dificultad Media.

16. 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 construid el vector pedido. Haced lo mismo pero a la inversa, es decir, construir la matriz a partir del vector. En este ejercicio, no hay que construir ninguna clase ni función. Es un ejercicio sobre recorridos de una matriz.

Dificultad Media.

RELACIÓN DE PROBLEMAS V. Clases (Segunda parte)

Problemas Básicos

1. Cread la clase RLE para codificar/descodificar una secuencia según indica el ejercicio 22 de la Relación de Problemas II de Estructuras de Control. Los métodos a implementar tendrán las siguientes cabeceras:

```
MiVectorEnteros Codifica      (MiVectorEnteros
                               secuencia_descodificada)
MiVectorEnteros Descodifica (MiVectorEnteros
                               secuencia_codificada)
```

Por ejemplo, la llamada al método `Codifica()` pasándole como parámetro el vector (1 1 1 2 2 2 2 2 3 3 3 3 3 5) debe devolver el vector (3 1 5 2 6 3 1 5) mientras que el método `Descodifica()` hace lo contrario.

Escribid un programa principal que lea un número indeterminado de enteros (terminando con un -1) y los guarde en un objeto de la clase `MiVectorEnteros`. A continuación la codificará y descodificará en una nueva secuencia.

Añadir un método público a la clase `MiVectorEnteros`:

```
bool EsIguala (MiVectorEnteros otro);
```

que sirva para comparar dos datos de tipo `MiVectorEnteros`: serán iguales si contienen el mismo número de elementos, son iguales y están en el mismo orden. Usad este método para demostrar que la secuencia original es igual a la secuencia obtenida después de codificarla y descodificarla.

Finalidad: Ver métodos a los que se les pasa como parámetro y devuelven un objeto de otra clase. Dificultad Baja.

2. Usad la clase `Punto2D` para representar un punto en \mathbb{R}^2 y cread una clase llamada `Circunferencia` que represente el centro con un dato del tipo `Punto2D` (recordad el ejercicio 33 de la Relación de Problemas III).

Añadid métodos para:

- a) obtener la longitud de la circunferencia,
- b) obtener el área del círculo interior,
- c) obtener el valor del centro de la circunferencia,
- d) obtener el valor del diámetro de la circunferencia,

RELACIÓN DE PROBLEMAS V. Clases (Segunda parte)

- e) saber si la circunferencia contiene a un punto. Un punto (x, y) está dentro de una circunferencia con centro (x_0, y_0) y radio r si se verifica que:

$$(x_0 - x)^2 + (y_0 - y)^2 \leq r^2$$

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

Finalidad: Declarar un objeto como dato miembro de otro objeto. Dificultad Baja.

3. Supongamos que tenemos una clase para generar números enteros aleatorios entre un mínimo y un máximo, con la siguiente interfaz pública:

```
MyRandom
+ MyRandom(int minimo, int maximo)
+ int Next()
+ int Min()
+ int Max()
```

Puede usarse cualquiera de las implementaciones que vienen a partir de la página [RP-V.15](#). Para generar 10 números aleatorios entre 4 y 7, por ejemplo, bastaría hacer lo siguiente:

```
MyRandom generador_aleatorio(4, 7);

for (int i=0; i<10; i++)
    cout << generador_aleatorio.Next();
```

Hay que destacar lo siguiente:

- Cada llamada a `generador_aleatorio.Next()` genera un valor aleatorio (entre 4 y 7 en el ejemplo)
- Los valores generados pueden repetirse antes de que se hayan generado todos los posibles valores. Por lo tanto, una posible secuencia de números generados podría ser la siguiente: 5 4 5 6 4 7 4 5

Se pide crear la clase `GeneradorPermutaciones` para generar permutaciones aleatorias de un conjunto de enteros entre un valor mínimo y un valor máximo. La clase tendrá un único método con la siguiente cabecera:

```
Permutacion Genera(int primero, int ultimo)
```

dónde la clase `Permutacion` es la vista en el problema [12](#) de la Relación de Problemas IV. Por ejemplo, si mínimo = 1 y máximo = 6, una permutación válida sería

RELACIÓN DE PROBLEMAS V. Clases (Segunda parte)

{3,1,6,4,5,2}. Como puede observarse, no pueden aparecer elementos repetidos y deben estar todos los valores entre 1 y 6.

Finalidad: Métodos que devuelven objetos y usan localmente otros objetos. Dificultad Media.

4. Reescribir la solución al ejercicio 3 de esta relación de problemas (generador de permutaciones aleatorias) usando como dato miembro de la clase `Permutacion` un objeto de la clase `MiVectorEnteros`, en vez de un vector de la STL. Al constructor le seguimos pasando un vector de la STL. Reemplazad los métodos privados que había definidos dentro de `Permutacion` (como por ejemplo el método `PosMax()`) por las correspondientes llamadas a los métodos de `MiVectorEnteros`.

Finalidad: Trabajar con un objeto como dato miembro de otro objeto. Dificultad Baja.

5. (*Examen Septiembre 2012*) Definid la clase `ConjuntoOrdenado` para que permita almacenar una secuencia **ordenada** de números enteros **sin repetidos**. Definid métodos para:

- Añadir un entero (de forma ordenada y sin almacenar repetidos).
- Calcular la unión con otro conjunto. En la unión se deben incluir los elementos que estén en cualquiera de ellos.
- Calcular la intersección con otro conjunto. En la intersección se deben incluir los elementos que sean comunes a ambos conjuntos.

6. Recuperad la solución al problema 11 del ComeCocos de la Relación de Problemas IV. Sobre la clase `CaminoComeCocos`, añadidle un método que compruebe si el conjunto de movimientos de un camino contiene a los movimientos de un segundo camino que se pasará como parámetro al método. Debe respetarse el orden en el que aparecen los movimientos, pero no tienen por qué estar consecutivos. Por ejemplo, el camino {'s','s','b','i','d','d'} contiene al camino {'s','i'} pero no al camino {'i','s'}. Puede usarse la sobrecarga del método `find` de la clase `string` a la que se le pasa como parámetro la posición inicial desde la que se realiza la búsqueda:

```
string cadena = "Hola, soy yo";
int pos;
pos = cadena.find('o',2); // 7 -> Primera aparición de 'o'
                        //      a partir de la posic. 2
pos = cadena.find('o',1); // 1
pos = cadena.find('H',1); // -1 -> No encontrado.
```

Finalidad: Pasar a un método de una clase un parámetro de la propia clase. Dificultad Baja.

7. Sobre la clase `MiVectorCaracteres` (la que usa un vector “clásico” como dato privado) definid el método `Contiene` para ver si contiene a otro vector de caracteres. Los caracteres tienen que estar consecutivos y en el mismo orden. Por ejemplo, el vector `{'t','t','b','i','d','d'}` no contiene al vector `{'t','i'}` pero sí contiene al vector `{'t','b','i'}` (a partir de la posición 1).

Nota: Si este ejercicio hubiese que hacerlo sobre la clase `CaminoComeCocos` sería muy fácil ya que bastaría llamar a la sobrecarga del método `find` de la clase `string` a la que se le pasa como parámetro otro `string`:

```
string cadena = "Hola, soy yo";  
int pos;  
pos = cadena.find("soy");    // 6
```

En este ejercicio se pide que se implemente **explícitamente** la búsqueda dentro de `MiVectorCaracteres`, por lo que **no** se puede usar la clase `string`.

Finalidad: Pasar a un método de una clase un parámetro de la propia clase. Dificultad Media.

8. Considerar la clase `ColeccionPuntos2D` que se usará para almacenar y gestionar una colección de datos de tipo `Punto2D`. Por simplicidad consideraremos únicamente puntos cuyas coordenadas sean positivas.

Realizar un programa que lea del teclado un número indeterminado de datos de tipo `Punto2D` de manera que termine la lectura si el usuario escribe `-1` cuando el programa le pide la abscisa de un nuevo punto. Los puntos leídos los almacena en un objeto `ColeccionPuntos2D`.

A continuación pide los datos necesarios, y crea un objeto `Circunferencia`, y finalmente muestra cuáles de los puntos almacenados en la colección `ColeccionPuntos2D` está en el círculo delimitado por la circunferencia.

9. Cread la clase `ImpresorVectorEnteros` para imprimir los datos de un objeto de la clase `MiVectorEnteros` (la que usa un vector “clásico” como dato privado).

Finalidad: Ver métodos a los que se les pasa como parámetro un objeto de otra clase. Dificultad Baja.

10. Cread la clase `LectorVectorEnteros` que sirva para leer los datos de un objeto de la clase `MiVectorEnteros` (la que usa un vector “clásico” como dato privado).

La lectura de datos parará cuando se llegue a un terminador, que será un valor entero especial que habrá que especificarlo de alguna forma.

Finalidad: Ver métodos que devuelven un objeto de otra clase. Dificultad Baja.

11. En clase de teoría se ha definido la clase `MiMatrizCaracteres` para representar una matriz, en la que cada fila puede tener un número distinto de columnas. En la página [RP-V.19](#) puede encontrarse el código de esta clase.

RELACIÓN DE PROBLEMAS V. Clases (Segunda parte)

Cread un programa principal en el que se cree una matriz, se le añadan varias filas y se impriman todos sus caracteres. Para imprimir los caracteres, accederemos en primer lugar a cada fila de la matriz a través del método `Fila()`. Imprimiremos cada objeto devuelto por `Fila()` de dos formas alternativas:

- Con un bucle que vaya recorriendo los caracteres de dicha fila a través del método `Elemento()`.
- Usando la clase `ImpresorVectorCaracteres`, similar a la vista en el ejercicio 9 de esta relación de problemas.

Finalidad: Usar matrices desde main. Dificultad Baja.

- Ampliad el ejercicio 11 definiendo una clase `ImpresorMatricesCaracteres` para que imprima matrices de caracteres. En la definición de esta clase, utilizad la clase `ImpresorVectorCaracteres`, basada en la clase vista en el ejercicio 9 de esta relación de problemas.

Finalidad: Pasar a un método de una clase un parámetro de otra clase. Dificultad Baja.

- (Examen Febrero 2011) Añadid un método a la clase `MiMatrizCaracteres` para que inserte una fila completa en una posición (de fila) determinada. Por ejemplo,

$$\begin{pmatrix} c & b & a & l & ? \\ f & g & h & ? & ? \\ ? & ? & ? & ? & ? \end{pmatrix} \xrightarrow[\text{en la fila 1}]{\text{Insertar } (d, d, e, e)} \begin{pmatrix} c & b & a & l & ? \\ d & d & e & ? & ? \\ f & g & h & i & ? \end{pmatrix}$$

dónde ? representa un valor no utilizado.

Finalidad: Pasar a un método de una clase un parámetro de otra clase. Dificultad Baja.

- Cread la clase `MiMatrizRectangularCaracteres` para obligar a que todas las filas tengan el mismo número de columnas, es decir, que sea una matriz rectangular. Usad como dato miembro privado una matriz clásica en la forma:

```
char matriz_privada[MAXIMO_FILAS][MAXIMO_COLUMNAS];
```

Definid métodos para:

- Obtener el número de filas y columnas utilizadas.
- Obtener el carácter que haya en una fila y columna.
- Devolver una fila completa como un vector de la STL.
- Añadir una fila entera. La fila será un vector de la STL.
- Encontrar un carácter.
- Comprobar si es igual a otra matriz rectangular.
- (Examen Febrero 2011) Obtener la traspuesta de la matriz.

Finalidad: Trabajar con matrices. Dificultad Baja.

15. Sobre la clase `MiMatrizRectangularCaracteres` del ejercicio anterior, añadir un método para comprobar si es simétrica. Hacedlo primero calculando la traspuesta de la matriz y viendo si es igual a su simétrica, usando los métodos de los ejercicios anteriores. Hacedlo también comprobando directamente si cada componente es igual a su *simétrica* y parando el recorrido en cuanto encuentre una componente que no lo verifique.

Finalidad: Trabajar con matrices. Dificultad Media.

16. Definid la clase `MiMatrizRectangularEnteros` de forma análoga a la clase `MiMatrizRectangularCaracteres` (ejercicio 14 de esta relación de problemas). Ampliar el conjunto de métodos básicos con otros que nos permitan realizar las siguientes operaciones:

- a) (*Examen Febrero 2011*) Sumar aquellos valores M_{ij} de la matriz M que verifiquen que $i + j$ sea igual a:

$$\sum_{h=1}^k h^2 \quad \text{para algún valor de } k \geq 1$$

- b) (*Examen Septiembre 2011*) Calcular 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 está en (0, 2).

- c) Ver si existe un valor *MaxiMin*, es decir, que sea a la vez, máximo de su fila y mínimo de su columna.
17. Ampliar la clase `MiMatrizRectangularEnteros` con un **método** que busque la fila de la matriz que más se parezca a un vector de enteros (clase `MiVectorEnteros`) al que llamaremos *referencia*. El método devolverá el *número* de la fila.
- La similitud entre dos vectores $x = (x_1 \cdots x_p)$ e $y = (y_1 \cdots y_p)$ vendrá dada por la distancia euclídea entre ambos:

$$dist(x, y) = \sqrt{(x_1 - y_1)^2 + \cdots + (x_p - y_p)^2}$$

Además, la búsqueda solo se hará sobre las **filas** de la matriz enumeradas en un segundo vector de enteros (clase `MiVectorEnteros`) llamado `filas_a_comparar`.

RELACIÓN DE PROBLEMAS V. Clases (Segunda parte)

Por ejemplo, dada la matriz M (7×4),

→	3	1	0	8
	4	5	1	5
→	5	7	1	7
	7	9	6	1
→	4	9	5	5
→	2	8	2	2
	7	3	2	5

y los vectores `referencia = 2,8,1,1` y `filas_a_comparar = 0,2,4,5`, el programa deberá encontrar 5 como la fila más cercana al vector `referencia` (en el dibujo anterior se han marcado con una flecha las filas indicadas por el vector `filas_a_comparar`).

Problemas de Ampliación

18. Según un estudio de una universidad ignifera, no importa el orden en el que las letras están escritas, la única cosa importante es que la primera y la última letra estén escritas en la posición correcta. El resto pueden estar totalmente mal y aún podrías leerlo sin problemas. Esto es porque no leemos cada letra por sí misma sino la palabra como un todo.

Se pide crear la clase `Palabra` que permita almacenar un conjunto de caracteres que representarán una palabra. Definid un método `EsIgual` al que se le pase como parámetro otra palabra y determine si son iguales atendiendo al siguiente criterio: La primera letra de ambas palabras es igual, la última letra de ambas palabras también es igual y el resto de las letras son las mismas pero no están necesariamente en las mismas posiciones.

Dificultad Baja.

19. Reescribid la solución al ejercicio 15 de la relación de problemas IV (Vectores) para que use como dato miembro privado un vector de palabras. Cada palabra será un objeto de la clase `Palabra`.

Dificultad Media.

20. (*Examen Septiembre 2012*) Se quiere desarrollar una aplicación para automatizar la realización de exámenes tipo test. El software incluirá una clase `Examen` que debe almacenar: el nombre de la asignatura, la lista de enunciados de las preguntas (cada enunciado es una cadena de caracteres de tipo `string`) y la lista de respuestas correctas para cada pregunta (cada respuesta es un carácter). Para representar una lista puede usar un vector de la STL. Implementa la clase junto con los siguientes métodos:

- Un constructor que inicialice un objeto de tipo `Examen` dando el nombre de la asignatura y con la lista de preguntas vacía.
- Un método `NuevaPregunta` que reciba un enunciado y la respuesta correcta y que los añada a la lista de preguntas del examen. Cada nueva pregunta siempre se añade al final de la lista.
- Un método `NumPreguntas` que devuelva el número de preguntas de que consta el examen.
- Un método `GetEnunciado` que devuelva el enunciado de la pregunta *i*-ésima.
- Un método `GetRespuesta` que devuelva la respuesta de la pregunta *i*-ésima.

A continuación, se pide realizar un programa que permita evaluar a una serie de alumnos utilizando la clase `Examen`. El programa comenzará creando un objeto de tipo `Examen` y dándole contenido, es decir, leyendo las preguntas y respuestas correctas desde la entrada estándar y almacenándolas.

RELACIÓN DE PROBLEMAS V. Clases (Segunda parte)

Una vez leído el examen se procederá a la evaluación de un número de alumnos dado desde la entrada estándar. Para ello el programa le mostrará las preguntas del examen a cada alumno y leerá sus respuestas. Al finalizar cada alumno la prueba, el programa le dirá su nota de acuerdo a los siguientes criterios:

- Por cada pregunta sin responder se suman 0 puntos.
- Por cada respuesta correcta se suma 1 punto.
- Por cada respuesta incorrecta se resta 1 punto.
- La nota final estará en el intervalo [0, 10]. Un 10 significa que ha respondido y acertado todas las preguntas. Si la calificación es negativa se sustituye por cero.

No es necesario almacenar las notas de los alumnos ya que se pueden ir mostrando al terminar cada uno de ellos la prueba. Además, se pueden añadir nuevos métodos a la clase `Examen` si lo considera oportuno, así como implementar funciones externas a la misma.

Dificultad Baja.

21. 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'. El método devolverá un `struct` del tipo:

```
struct FrecuenciaCaracter{
    char caracter;
    int  frecuencia;
}
```

en el que el campo `caracter` contendrá el carácter en cuestión ('o') y en el campo `frecuencia` el conteo de la moda (3).

Dificultad Media.

22. (*Examen Septiembre 2012*) Definid la clase `VectorParejasCaracterEntero` que permite almacenar un conjunto de parejas de la forma (`carácter`, `entero`).

Se pide crear un método que borre de un vector de caracteres (objeto de la clase `MiVectorCaracteres`) cada uno de los caracteres que aparecen en el vector de parejas, tantas veces como indique el entero asociado a ese carácter.

Cada pareja de `VectorParejasCaracterEntero` será un `struct ParejaCaracterEntero`

```
struct ParejaCaracterEntero{
    char caracter;
    int  veces;
}
```

en el que el campo `caracter` contendrá el carácter a borrar y el campo `veces` el número de ocurrencias que deben borrarse.

Por ejemplo:

Borrar $\{(a,1), (b,2)\}$ en $\{a,b,a,b,c,a,b,d,a\} \rightarrow \{a,c,a,b,d,a\}$

Finalidad: Trabajar con un vector de objetos. Dificultad Media.

23. (*Examen Septiembre 2009*) Sobre la clase `MiVectorCaracteres`, añadid un método que determine si dicha secuencia de caracteres $C1$ contiene a otra secuencia $C2$ en el mismo orden (no tienen que estar consecutivos) y de forma *cíclica*. Para que se cumpla este criterio se deben satisfacer las siguientes condiciones

- Todos los caracteres de $C2$ deben estar en $C1$
- Deben estar en el mismo orden aunque no de forma consecutiva
- Si durante la búsqueda se ha llegado al final de la secuencia $C1$, se debe proseguir la búsqueda por el inicio de $C1$, pero sin sobrepasar la posición en la que hubo la primera concordancia.

Se muestran algunos ejemplos en los que la secuencia $C1$ contiene a $C2$:

- $C1 = \text{xzayobnmcpwqdfg}$ $C2 = \text{abcd}$
- $C1 = \text{ftkcpxqdhjzaxqoblki}$ $C2 = \text{abcd}$
- $C1 = \text{tzsbluyclpygdmngrafrvc}$ $C2 = \text{abcd}$

Hay que destacar que la primera letra de $C2$ a buscar en $C1$ podría estar en cualquier sitio. Por ejemplo, para el siguiente caso:

- $C1 = \text{bghcjadxak}$ $C2 = \text{abcd}$

podemos ver que a partir de la primera a de $C1$ no podemos encontrar $C2$ de forma cíclica, aunque sí lo podemos hacer a partir de la segunda a de $C1$.

También puede darse el caso de que $C1$ no contenga a $C2$ de forma cíclica aunque incluya todas sus letras, como muestra el siguiente ejemplo:

- $C1 = \text{tzsbluyclpcaygdmxngrfvc}$ $C2 = \text{abcd}$

24. A la hora de diseñar una clase debemos poner especial énfasis en intentar que sea lo más reutilizable posible, intentando que su interfaz pública no esté ligada a los detalles de la implementación. Por ejemplo, en el ejercicio 17 de la relación de problemas III se definió una clase para leer una opción del usuario. Para ello, se definió una clase `Menu` con un método `LeeOpcion()` que devolvía un `char`. ¿Y si quisiésemos modificar el criterio de entrada y que la opción fuese un entero en vez de un carácter o que fuesen caracteres distintos? Habría que hacerlo en dos sitios: por un parte, dentro de la clase `Menu` y por otra parte en el programa `main` en el que se utiliza. Para evitarlo, podemos definir el método `LeeOpcion()` para que devuelva algo más genérico que un `char`, por ejemplo, un enumerado que represente la opción leída. De esta forma, si queremos cambiar la forma en la que se lee la opción (por ejemplo '1' en vez de 'P', '4' en vez de 'I', etc), bastará modificar la implementación interna de la clase `Menu`, pero los *clientes* de dicha clase (en nuestro caso el programa principal `main`) seguirán utilizándola de la misma forma, a través del enumerado que devuelve el método.

Recuperad la solución al ejercicio 17 disponible en la web del departamento y realizad la transformación pedida.

Finalidad: Enfatizar la necesidad de abstraer la interfaz de una clase. Dificultad Baja.

25. En el ejercicio 21 de la relación de problemas III se creó una clase para representar una fracción. Recuperad la solución disponible en la web de la asignatura. Se pide rediseñar la clase `Fraccion` para que, en vez de trabajar con dos datos miembro `numerador` y `denominador`, trabaje con un único dato miembro que será un objeto del tipo `ParejaEnteros` (esta clase se vio en el ejercicio 16 de la relación de problemas III). Se pide:

- Definid dos constructores para la clase `Fraccion`:
 - Uno de ellos se le pasará como parámetros dos enteros que representen el numerador y el denominador (aunque se le pasen dos enteros, hemos indicado que la clase contendrá un único dato miembro privado de tipo `ParejaEnteros`)
 - Al otro constructor se le pasará un objeto de la clase `ParejaEnteros`.
- La clase `ParejaEnteros` debe proporcionar el método `MCD` para calcular el máximo común divisor.
- Añadir también dos métodos para sumar y multiplicar dos fracciones. La declaración será de la forma

```
class Fraccion{
    .....
    Fraccion Suma(Fraccion otra_fraccion)
};
```

RELACIÓN DE PROBLEMAS V. Clases (Segunda parte)

- Cread un programa principal que cree varios objetos fracciones y ejecute sus métodos.

Finalidad: Declarar un objeto como dato miembro de otro objeto y pasar como parámetro un objeto de la misma clase. Dificultad Baja.

26. Se desea gestionar una librería. Para cada libro, únicamente se desea almacenar un identificador y la fecha de su publicación. Para ello, se pide definir las siguientes clases:

- La clase `Identificador` contendrá un dato miembro privado de tipo `string` en el que se almacenará un identificador de 4 caracteres. Proporcionará, al menos, sendos métodos `EsIgual_a()`, `EsMenor_que()` para comparar con otro objeto de la clase `Identificador`. Diremos que un objeto identificador es menor que otro según lo indique el orden lexicográfico de sus datos miembros de tipo `string`. Por ejemplo, la cadena "ABZZ" es menor que "ACAA" y "AAAB" es menor que "AAAC. Para comparar dos cadenas, usad el método `compare` de la clase `string`.

`cadena.compare(otra_cadena)` devuelve 0 si son iguales, un entero negativo si `cadena` es menor que `otra_cadena` y un positivo en caso contrario.

- La clase `Fecha` se construirá a partir del código proporcionado en la página [RP-V.17](#). Hay que añadirle sendos métodos `EsIgual_a()`, `EsMenor_que()` para comparar con otro objeto de la clase `Fecha`. Diremos que un objeto de fecha es menor que otro según lo indique el orden natural de las fechas correspondientes.
- La clase `Libro` contendrá como datos miembros un `Identificador` y una `Fecha`. Proporcionará, al menos, sendos métodos `EsIgual_a()`, `EsMenor_que()` para comparar con otro objeto de la clase `Libro`. Posteriormente se indica cual es el criterio de comparación entre dos libros.
- La clase `Biblioteca` contendrá un conjunto de libros. Puede usar como dato miembro privado un vector de `Libro`. Debe proporcionar al menos métodos para añadir y recuperar libros.
- La clase `LectorBiblioteca` para leer los datos desde un fichero y crear una biblioteca. Los datos estarán dispuestos de la siguiente forma: Una cadena de caracteres con el identificador y tres enteros con el día, mes y año de publicación. Habrá tantas repeticiones como libros haya. El final del fichero vendrá marcado por la aparición de la cadena "FFFF", como por ejemplo:

```
JFGT 30 1 2003 JGHT 26 2 1998 YFGT 30 1 2003 AUTQ 26 2 1998 FFFF
```

- La clase `ImpresorBiblioteca` para que imprima en pantalla todos los libros.

Se pide construir el método `Ordena()` sobre la clase `Biblioteca` para que ordene los libros por orden de fecha de menor a mayor. A igualdad de fechas, ordenar por

RELACIÓN DE PROBLEMAS V. Clases (Segunda parte)

orden del identificador de menor a mayor. En el ejemplo anterior, la biblioteca quedaría como sigue:

AUTQ 26 2 1998 JGHT 26 2 1998 JFGT 30 1 2003 YFGT 30 1 2003

- El algoritmo de ordenación es similar a cualquiera de los vistos en el tema IV. Simplemente cambia el tipo de dato del vector sobre los que se trabaja.
- Puede utilizar el proyecto BibliotecaEsqueleto disponible en la plataforma decsai que proporciona el código básico de alguna de las clases pedidas.

Cambiar ahora el método `EsMenor_que()` de la clase `Libro` para que ordene por orden de fecha de mayor a menor. A igualdad de fechas, ordenar por orden del identificador de menor a mayor. En el ejemplo anterior, la biblioteca quedaría como sigue:

JFGT 30 1 2003 YFGT 30 1 2003 AUTQ 26 2 1998 JGHT 26 2 1998

Finalidad: Trabajar con un vector de objetos. Dificultad Media.

Problemas sobre Excepciones

27. Como ya se ha comentado en clase de teoría, si se produce la violación de las precondiciones de un método podemos hacer varias cosas:

- El método no hará nada o devolverá un valor indeterminado.
El caso típico sería el método `Elemento` de la clase `MiVectorCaracteres`. Si se le pasa un índice incorrecto, devolvería cualquier valor.
- El método lanza una excepción para notificar el error producido.
El caso típico sería el método `Inserta` de la clase `MiVectorCaracteres`. Si se le pasa un índice incorrecto, el método lanza una excepción.
Nota: Si se estima oportuno, el mismo método anterior `Elemento` podría lanzar una excepción, aunque al ser un método muy usado, optamos por aumentar la eficiencia no comprobando que el índice sea correcto.

En la relación de problemas IV se trabajó con la clase `MiVectorCaracteres`. Analizar los métodos que deberían lanzar una excepción y programarlos, incluyendo un programa principal que capture dichas excepciones. Incluir, al menos, los métodos `Aniade`, `TotalUtilizados`, `Modifica`, `Elemento` y los que resolvían los ejercicios 1 (Palíndromo), 3 (Invierte), 4 (Elimina) y 5 (Elimina mayúsculas) de la relación de problemas IV.

Las excepciones que se lancen deben ser de la biblioteca estándar `stdexcept`.

Finalidad: Lanzar excepciones para notificación de errores. Dificultad Baja.

28. Modificad la solución del ejercicio 25 de esta relación de problemas para que incluya el lanzamiento de excepciones en aquellos sitios dónde haya que hacerlo.

Finalidad: Lanzar excepciones para notificación de errores. Dificultad Baja.

Apéndice: Clase MyRandom

(Disponible en <https://decsai.ugr.es> - *Material de la asignatura | Guión de Prácticas*)

Proporcionamos dos posibles implementaciones de la clase MyRandom para generar números enteros aleatorios entre un mínimo y un máximo.

Opción 1. Siguiendo el nuevo estándar de C++ 11.

```
// http://choorucode.wordpress.com/2010/11/24/
//      c-random-number-generation-using-random/

#include <random> // Hay que incluir estas bibliotecas
#include <ctime>

class MyRandom
{
private:
    typedef minstd_rand Engine;
    typedef uniform_int_distribution<int> Distribution;
    typedef variate_generator<Engine, Distribution> Generator;
    Generator gen;

public:
    MyRandom(int minVal, int maxVal )
        :gen( Generator( Engine( (unsigned)time(0) ),
            Distribution( minVal, maxVal ) ))
    {
        gen(); // desecho el primero
    }

    int Next(){
        return gen();
    }

    int Min(){
        return gen.min();
    }

    int Max(){
        return gen.max();
    }
};
```

RELACIÓN DE PROBLEMAS V. Clases (Segunda parte)

Opción 2. A la antigua usanza, para aquellos compiladores que no proporcionen la biblioteca random.

(Disponible en <https://decsai.ugr.es> - *Material de la asignatura | Guión de Prácticas*)

```
#include <cstdlib>    // Hay que incluir estas bibliotecas
#include <ctime>

class MyRandom
{
private:
    int minVal;
    int maxVal;

    void InitMyRandom (void)
    {
        time_t t;
        srand ((int) time(&t));    // Inicializa el generador
                                   // con el reloj del sistema
    }
public:
    MyRandom (int el_minimo, int el_maximo) :
        minVal(el_minimo), maxVal(el_maximo)
    {
        InitMyRandom();

        int no_lo_uso = Next(); // desecho el primero
    }

    int Next() {
        int rango = (maxVal - minVal)+1;
        int v1 = rango * (rand() / (RAND_MAX*1.0));
        int v2 = minVal + (v1 % rango);

        return v2;
    }

    int Min() {
        return minVal;
    }
    int Max() {
        return maxVal;
    }
};
```

Apéndice: Clase Fecha

(Disponible en <https://decsai.ugr.es> - *Material de la asignatura* | *Guión de Prácticas*)

```
class Fecha
{
private:

    unsigned int dia, mes, anio;

    /**/

    bool EsFechaCorrecta (unsigned int el_dia,
                          unsigned int el_mes,
                          unsigned int el_anio) {

        const unsigned int anio_inferior = 1900;
        const unsigned int anio_superior = 2500;
        const unsigned int dias_por_mes[12] =
            {31,29,31,30,31,30,31,31,30,31,30,31};

        bool es_dia_correcto, es_mes_correcto, es_anio_correcto;
        bool es_fecha_correcta;

        es_dia_correcto  = ((1 <= el_dia) &&
                           (el_dia <= dias_por_mes[el_mes - 1]));
        es_mes_correcto  = (1 <= el_mes && el_mes <= 12);
        es_anio_correcto = ((anio_inferior <= el_anio) &&
                           (el_anio <= anio_superior));
        es_fecha_correcta = es_dia_correcto && es_mes_correcto &&
                           es_anio_correcto;

        // Comprobación adicional sobre la propiedad "ser bisiesto"

        bool es_bisiesto = ((el_anio % 4 == 0) &&
                           (el_anio % 100 != 0)) || (el_anio % 400 == 0);

        if (es_fecha_correcta) &&
            if (el_mes == 2) && (el_dia == 29) && (!es_bisiesto))
                es_fecha_correcta = false;

        return (es_fecha_correcta);
    }
}
```

public:

```
    /**
    // Constructor con parámetros. Realiza una comprobación
    // exhaustiva de la validez de los datos recibidos
    // usando el método privado EsFechaCorrecta()

    Fecha (unsigned int el_dia,
           unsigned int el_mes,
           unsigned int el_anio) {

        // Si la fecha es correcta se inicializan los datos privados.
        // Si no lo fueran, se inician a valor 0.
        // Una solución mejor sería provocar el lanzamiento de una
        // excepción y de esta manera poder actuar adecuadamente.

        if (EsFechaCorrecta (el_dia, el_mes, el_anio)){
            dia = el_dia;
            mes = el_mes;
            anio = el_anio;
        }
        else
            dia = mes = anio = 0;
    }

    /**

    unsigned int GetDia()
    {
        return dia;
    }

    unsigned int GetMes()
    {
        return mes;
    }

    unsigned int GetAnio()
    {
        return anio;
    }
};
```

Apéndice: Clase Matriz de Caracteres

(Disponible en <https://decsai.ugr.es> - *Material de la asignatura* | *Guión de Prácticas*)

En esta clase se permite que cada fila tenga un número distinto de columnas.

```
class MiMatrizCaracteres
{
private:

    static const int MAXIMO_FILAS = 40;
    MiVectorCaracteres matriz_privada[MAXIMO_FILAS];
    int filas_utilizadas;

    // El número de columnas empleadas por cada fila, así como las
    // propiedades y los datos de cada fila se están disponibles
    // por los métodos de la clase MiVectorCaracteres

public:

    MiMatrizCaracteres() : filas_utilizadas(0)
    { }

    int FilasUtilizadas() {
        return filas_utilizadas;
    }

    char Elemento(int fila, int columna) {
        return (matriz_privada[fila].Elemento(columna));
    }

    MiVectorCaracteres Fila(int indice_fila)
    {
        return matriz_privada[indice_fila];
    }

    void Aniade(MiVectorCaracteres nueva_fila)
    {
        if (filas_utilizadas < MAXIMO_FILAS){
            matriz_privada[filas_utilizadas] = nueva_fila;
            filas_utilizadas++;
        }
    }

};
```

.

RELACIÓN DE PROBLEMAS VI. Recursividad

Problemas Básicos

1. Definir una función recursiva para sumar los dígitos de un entero positivo.
2. Cread sendas funciones recursivas para:
 - a) Calcular la división entera entre dos números enteros positivos.
 - b) Calcular el resto de la división entera de dos números enteros positivos.

Pasad como parámetros a cada función los dos enteros, y suponed que en la llamada siempre pasarán números positivos (es la precondition de la función)

En la definición de las funciones no pueden utilizarse los operadores de división, multiplicación ni módulo.

3. (*Examen Febrero 2012*) Sobre la clase `MiVectorCaracteres`, definir un método recursivo que nos diga si 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.

Pasad al método como parámetros las posiciones izquierda y derecha sobre las que trabajará.

4. Definir una función recursiva a la que se le pasen dos valores enteros n y k y devuelva como resultado el valor k -ésimo comenzando por la derecha del número n . Por ejemplo para $n = 427$ y $k = 3$ el resultado sería 4.
Si k es mayor que el número de dígitos de n la función devolverá el valor 0. Por ejemplo, para $n = 23$ y $k = 5$ el resultado es 0.
5. Definid una función recursiva con prototipo

```
int MCD(int un_entero, int otro_entero)
```

para que calcule el máximo común divisor entre dos enteros, aplicando el algoritmo de Euclides, que nos dice lo siguiente:

$$\begin{aligned} \text{MCD}(a, 0) &= a \\ \text{MCD}(a, b) &= \text{MCD}(b, a \% b) \end{aligned}$$

dónde $a \% b$ es el resto de la división entera entre a y b . Implementad el algoritmo suponiendo que a es mayor que b . A continuación, comprobad que también funciona en el caso contrario. ¿Por qué?

6. Cread una función recursiva con la siguiente cabecera

```
bool ContieneDesdeInicio(string cadena_grande, string cadena_peque)
```

que compruebe si la cadena `cadena_peque` se encuentra al inicio de la cadena `cadena_grande` (desde la posición cero)

Problemas de Ampliación

7. Cread una función recursiva con la siguiente cabecera

```
bool Contiene(string cadena_grande, string cadena_peque)
```

que compruebe si la cadena `cadena_peque` se encuentra dentro de la cadena `cadena_grande`. En la definición de esta función, se puede llamar a la función `ContieneDesdeInicio` del ejercicio 6

8. Definir una función recursiva que acepte un `__int64` y devuelva un vector de caracteres con los dígitos del mismo incluyendo los puntos de separación cada tres cifras decimales. Por ejemplo, para el entero 23456789, la función devolvería el vector de caracteres 23.456.789.

9. Definir una función recursiva para construir un vector de 0 y 1 con la representación en binario de un entero. En la página web

http://es.wikipedia.org/wiki/Sistema_binario

se puede consultar cómo pasar un número en decimal a binario. (el mismo procedimiento para pasar un decimal a binario se aplica sobre cualquier otra base menor o igual que 10).

El bit más significativo (el de la mayor potencia de 2) debe ir en la posición 0 del vector, mientras que el menos significativo (el correspondiente a 2^0) debe ir en la última posición del vector.

La función debe tener la siguiente cabecera:

```
MiVectorEnteros PasaBinario(int n)
```

Analizad la eficiencia de esta función.

10. Sobre la clase `MiVectorCaracteres`, definir un método recursivo que devuelva la diferencia entre el número de letras mayúsculas y el número de letras minúsculas que hay entre dos posiciones del vector. Por ejemplo, si el vector tuviese los caracteres `Hay UnAS LeTRas DESpuEs`, la diferencia entre mayúsculas y minúsculas entre las posiciones 4 y 15 sería $6 - 4 = 2$.

11. El método de *bisección* usado para calcular de forma aproximada el punto de corte de una función f con el eje de abscisas (la raíz de la función) se puede enunciar como sigue:

Sea $f(x)$ una función real, estrictamente monótona en $[i, d]$, donde $f(i)$ y $f(d)$ tienen distinto signo y sea ϵ una constante real pequeña (del orden de 10^{-4} , por ejemplo).

- a) Calcular m , el punto medio entre i y d ,
- b) Si $|i - d| < \epsilon$, terminar.
- c) Si $f(m)$ tiene igual signo que $f(i)$, repetir considerando el intervalo $[m, d]$
- d) Si $f(m)$ tiene igual signo que $f(d)$, repetir considerando el intervalo $[i, m]$

Implemente una función recursiva que reciba como datos de entrada los extremos i, d del intervalo, así como el valor de ϵ y devuelva la raíz de la función. *Se supone que la función $f(x)$ ya está definida con anterioridad.*

12. Realizar una función recursiva para multiplicar dos enteros según el método de multiplicación rusa (ver ejercicio 40 de la Relación de Problemas II (página RP-II.1)).
13. (*Examen Septiembre 2012*) Construir una función recursiva que muestre en la salida estándar los m mayores números pares que sean menores o iguales que un número n . El prototipo sería el siguiente:

```
void MayoresPares(int m, int n);
```

Por ejemplo:

- `MayoresPares(3,6)`: imprimirá en pantalla 6, 4, 2
- `MayoresPares(4,3)`: imprimirá en pantalla 2, 0, -2, -4
- `MayoresPares(0,8)`: no imprimirá nada en pantalla

.