

Práctica 1

Entorno de programación Microsoft Visual Studio.

Librerías estáticas y dinámicas

Material: PC y Microsoft Visual Studio 2013

Duración: 3 horas (1 hora clase de teoría + 2 horas prácticas)

Lugar: Laboratorios de prácticas (Laboratorio de Redes-Hardware)

La herramienta de desarrollo que utilizaremos para el desarrollo de las prácticas de la asignatura será el entorno de programación Microsoft Visual Studio 2013, el cual permite la creación, compilación y ejecución de programas escritos en lenguaje C; tanto en modo consola como en modo gráfico. Se recuerda al alumnado que, para la mejor comprensión y asimilación de los conocimientos y habilidades desarrollados durante la práctica, deberá haberse estudiado previamente el material docente disponible.

Introducción:

Un ordenador solamente es capaz de interpretar y ejecutar programas escritos en binario (lenguaje máquina). Un traductor es un programa que toma como entrada un programa escrito en un lenguaje de programación distinto al binario (código o lenguaje fuente) y lo transforma en un programa equivalente en binario (código o lenguaje objeto).

Antes de la ejecución de un programa es necesario que éste se encuentre libre de errores. Microsoft Visual Studio nos informa sobre el número y tipo de errores que hay en nuestro programa, para que los corrijamos y se pueda ejecutar sin problemas. Sin embargo, hay que tener en cuenta que, aunque el programa esté libre de errores, funcione y muestre un resultado, no quiere decir que su funcionamiento y/o comportamiento sea el correcto. Si esto

último ocurre, Microsoft Visual Studio nos proporciona una herramienta para averiguar las instrucciones que están provocando el fallo o el comportamiento anómalo del programa. A esta herramienta se le denomina depurador (del inglés debugger).

En esta práctica, repasaremos conceptos vistos en la asignatura de primer curso Informática: uso del entorno de programación Microsoft Visual Studio, incluyendo el manejo del debugger.

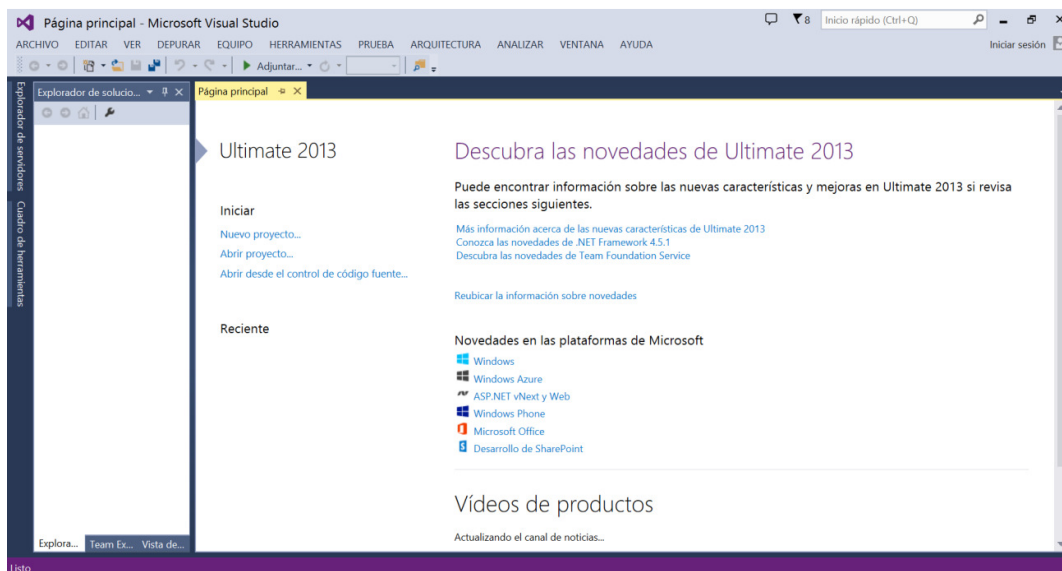
Desarrollo de la práctica:

1. Entorno de programación Visual Studio.

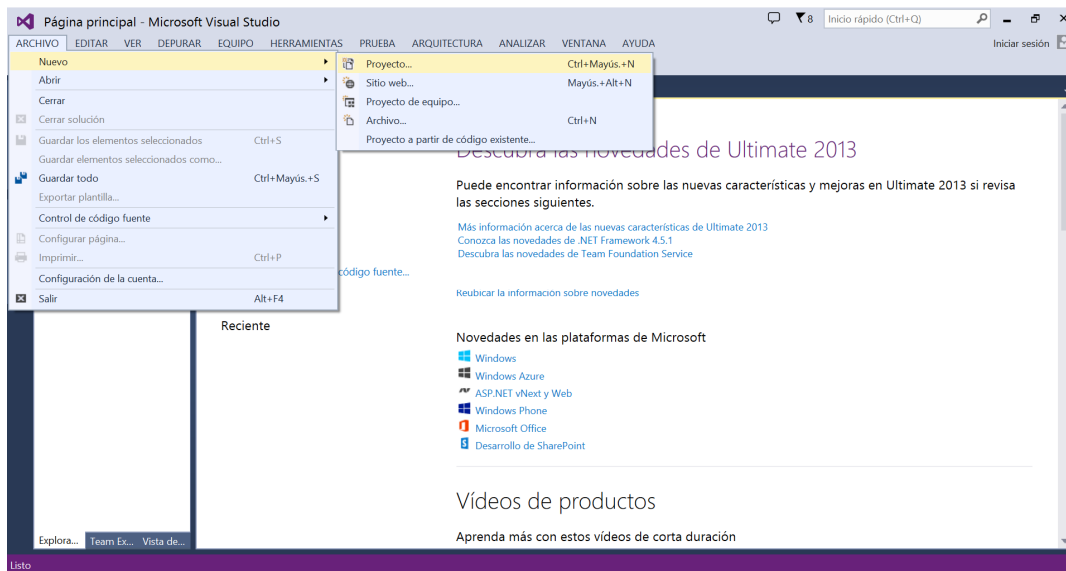
En esta primera parte de la práctica escribiremos un programa en C++ que visualice en la pantalla el mensaje "Hola mundo!!!". Para ello debemos crear un proyecto y un fichero donde guardar el código C/C++ del programa.

1) Creamos una carpeta en el Escritorio de nuestro PC con un nombre en concreto, en este ejemplo le pondremos de nombre a dicha carpeta *Práctica 1*. De esta forma, podremos identificar dónde se encuentran los archivos que el entorno de programación generará para el programa que desarrollemos.

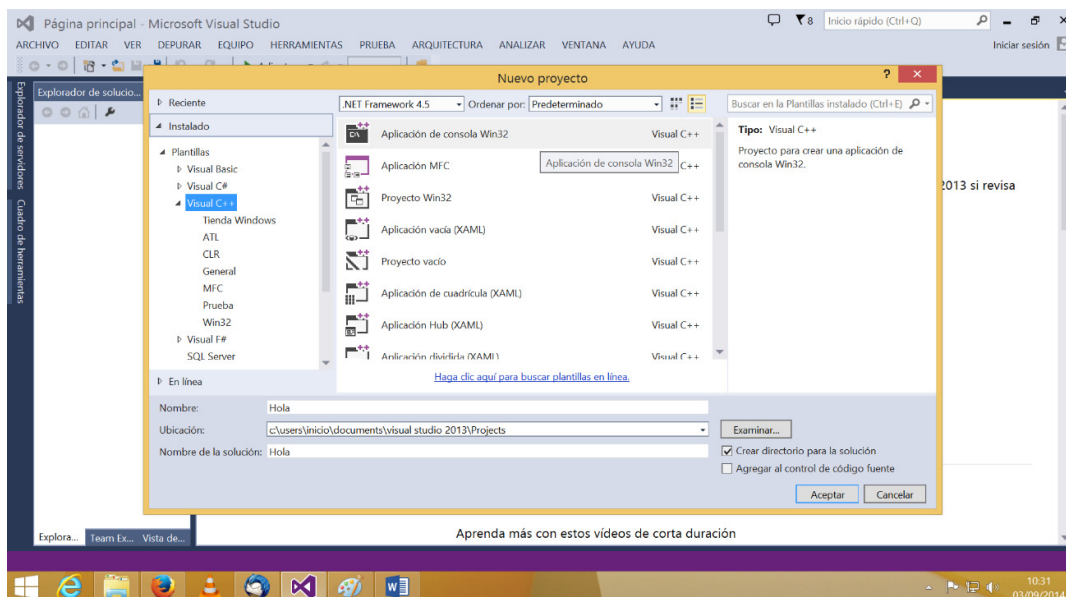
2) Ejecutamos el programa Visual Studio 2013, tras lo cual se nos mostrará la siguiente pantalla (o una similar, dependiendo de diversos factores).



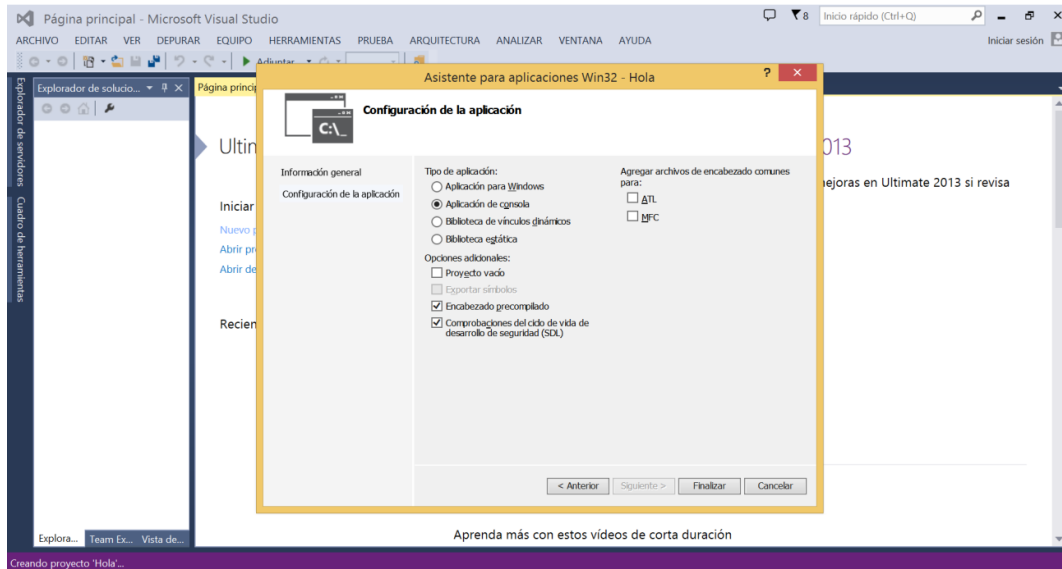
3) Seleccionaremos Archivo => Nuevo => Proyecto



4) En el menú de plantillas de la parte de la izquierda, que aparece en la ventana titulada como *Nuevo Proyecto*, comprobaremos que se encuentran seleccionadas las plantillas de C++, elegimos *Aplicación de consola Win32* como tipo de proyecto, ponemos *Hola* como nombre del proyecto que estamos creando (donde pone *Nombre*), en *Ubicación* indicaremos la carpeta donde queremos guardar nuestro proyecto (pulsando en *Examinar...* localizaremos la carpeta Práctica 1 que hemos creado en el escritorio en el paso 1) y, finalmente, pulsaremos Aceptar.



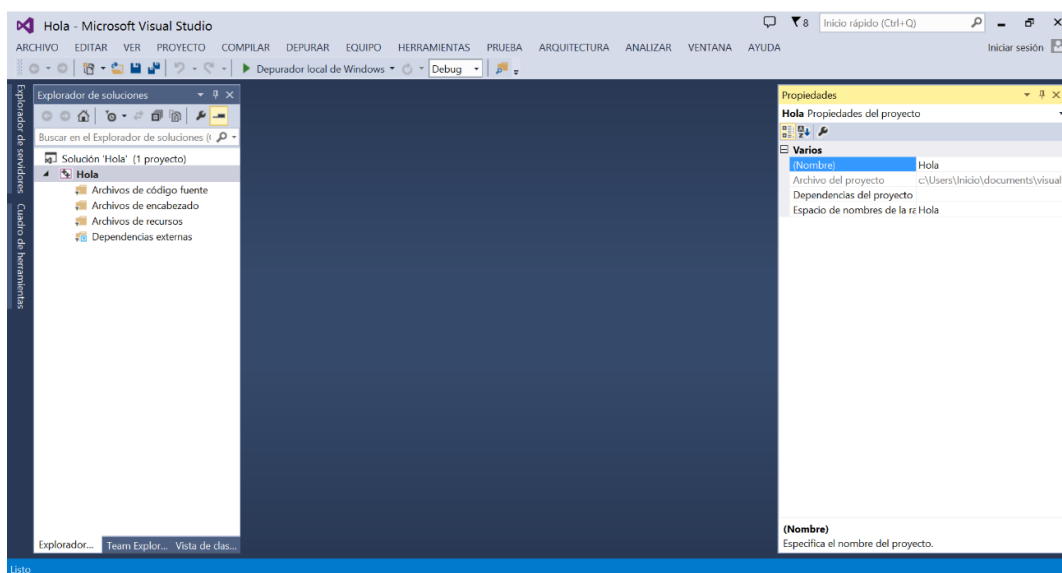
5) En la siguiente ventana que nos aparecerá, *Asistente para aplicaciones Win32*, pulsaremos en la opción *Siguiente*, apareciendo en ese momento la siguiente ventana:



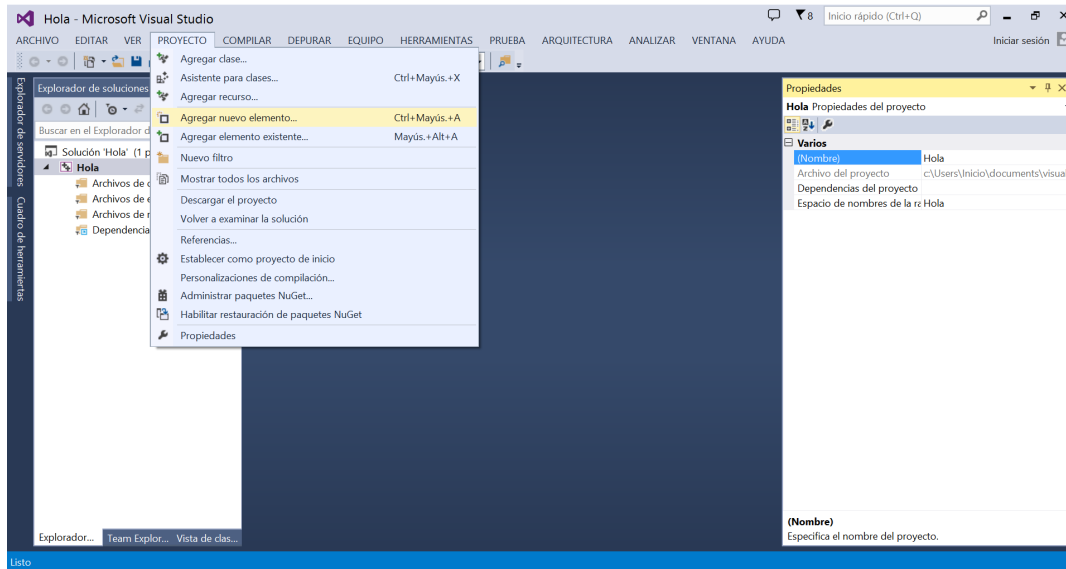
Dentro del apartado de Opciones adicionales debemos:

1. Desmarcar las opciones de *Encabezado precompilado* y de *Comprobaciones del ciclo de vida de desarrollo de seguridad (SDL)*.
2. Seleccionar la opción de *Proyecto vacío*.

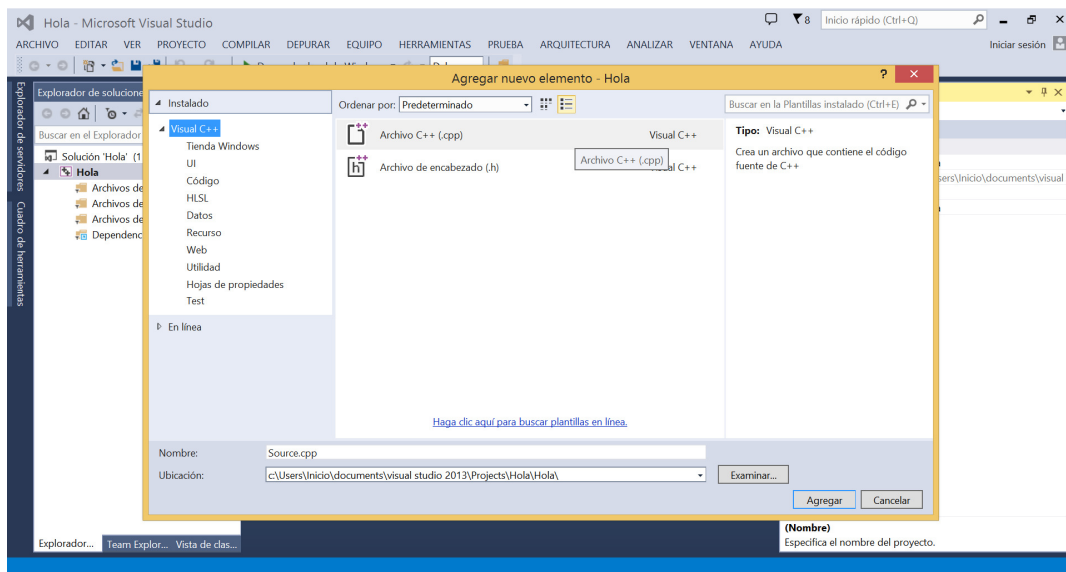
Finalmente, pulsaremos el botón de *Finalizar*, y el entorno mostrará la siguiente apariencia (o similar, dependiendo de los menús de información activos):



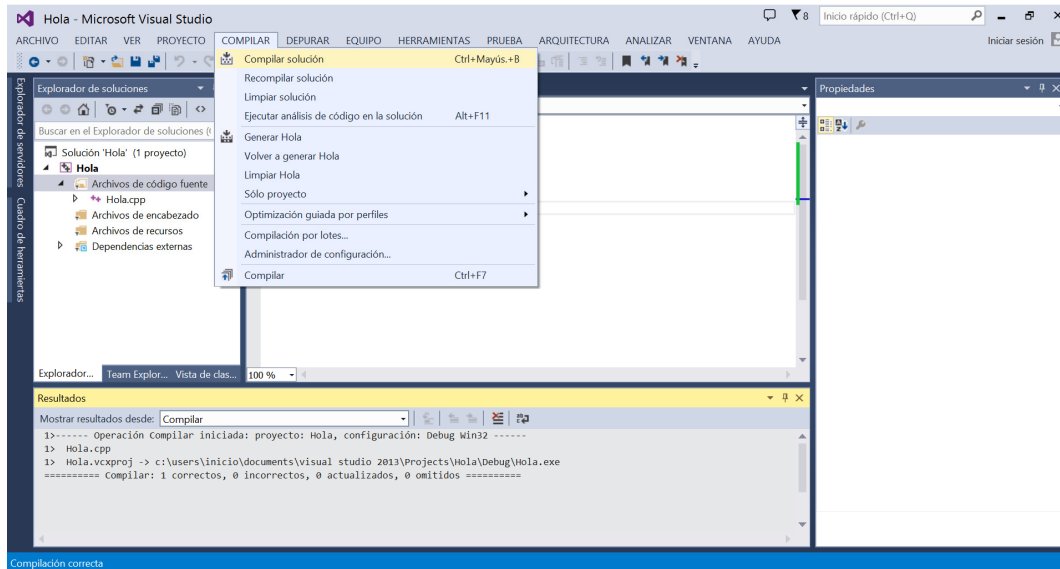
6) Es ahora cuando debemos añadir un fichero al proyecto para poder escribir el código del programa en él. Para ello, seleccionaremos la opción *Proyecto => Agregar nuevo elemento*.



7) Como nuevo elemento elegiremos Archivo C++ (.cpp) y como nombre *Hola.cpp*. Pulsaremos finalmente el botón *Agregar*. De esta forma ya podemos comenzar la escritura del programa.

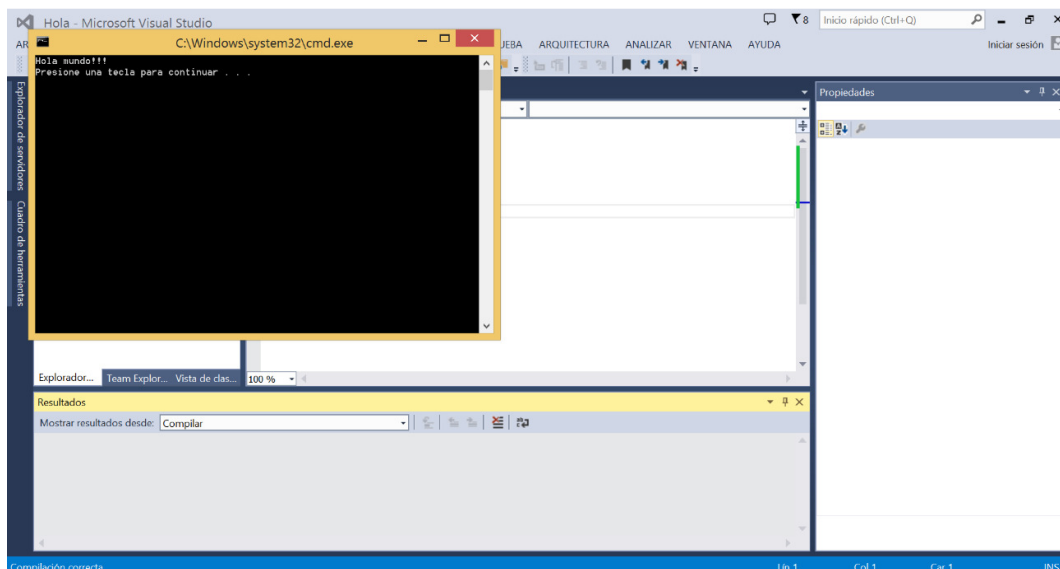


8) Escribimos el código del programa y lo compilamos para detectar posibles errores. Para ello, seleccionamos en la parte superior la opción *Compilar*, y a continuación *Compilar solución*.



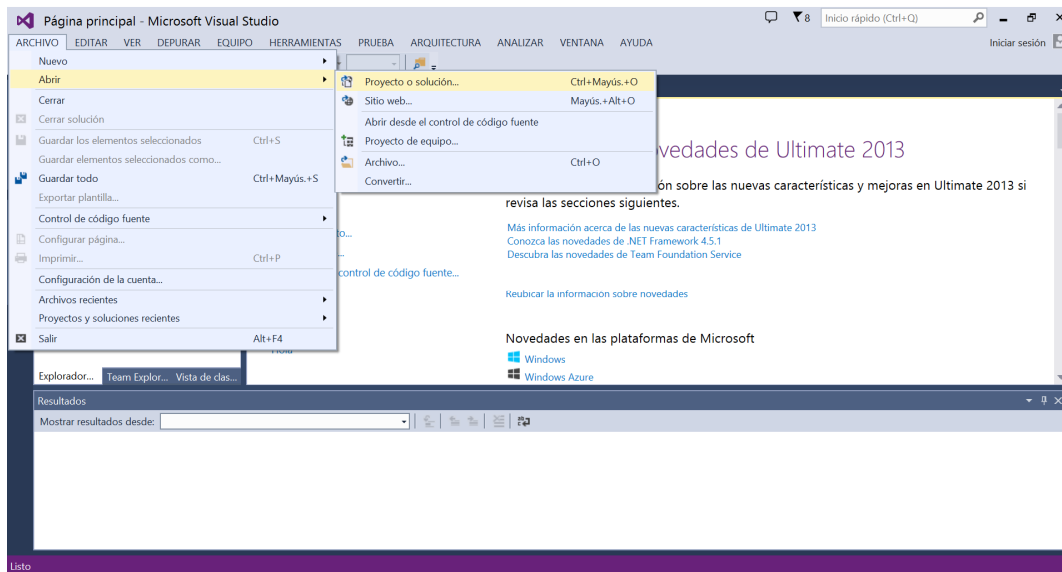
Si el código es correcto, la ventana de Resultados nos informará de ello.

9) Una vez tengamos el código libre de errores podremos ejecutarlo para ver el resultado. Para ello, seleccionamos en la parte superior la opción *Depurar* y a continuación *Iniciar sin depurar* (tecla rápida asociada Ctrl+F5). El resultado será el que se muestra a continuación:

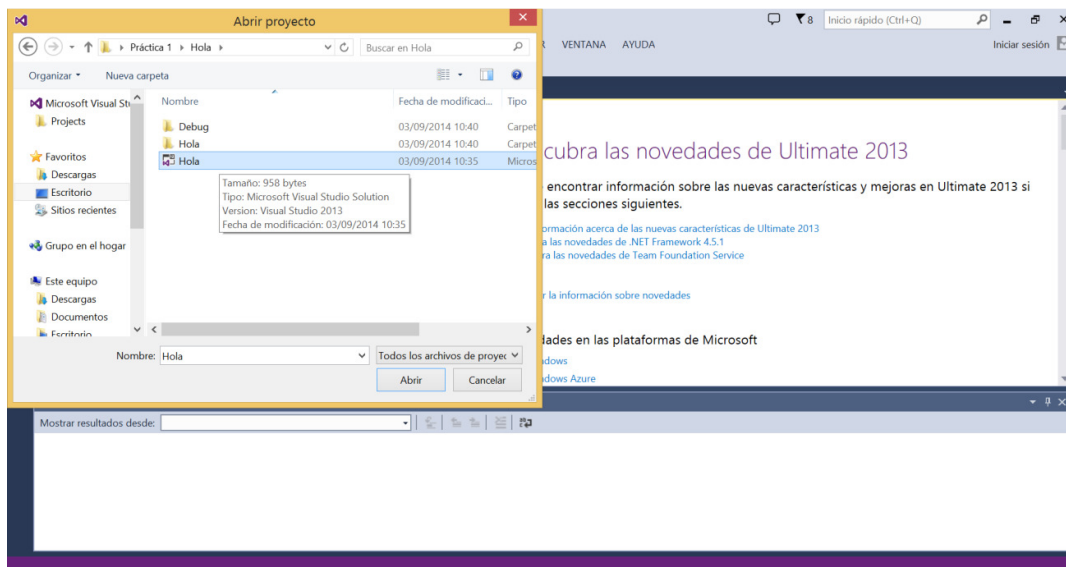


Para abrir un proyecto ya creado los pasos a seguir son los siguientes:

1) Seleccionar la opción Archivo => Abrir => Proyecto o solución.



2) Accederemos a la carpeta en la cual fue guardado el proyecto. Por ejemplo, si queremos abrir el proyecto creado en la primera parte de esta práctica, éste se encontrará dentro de la carpeta que creamos en el escritorio. Seleccionaremos el archivo maestro del proyecto y pulsaremos sobre el botón Abrir.



Tras estos pasos tendremos abierto nuestro proyecto y podremos continuar trabajando con él.

Para finalizar esta primera parte de la práctica, mediante el ejercicio propuesto número uno, repasaremos el empleo del depurador de entorno Microsoft Visual Studio.

Ejercicios:

1) Dado el siguiente programa escrito en C:

```
#include <stdio.h>

void main(void){

    //Ejemplos de uso de printf
    int v1=45;

    float numero1 = 125.738
    double resultado;
    char c="V";

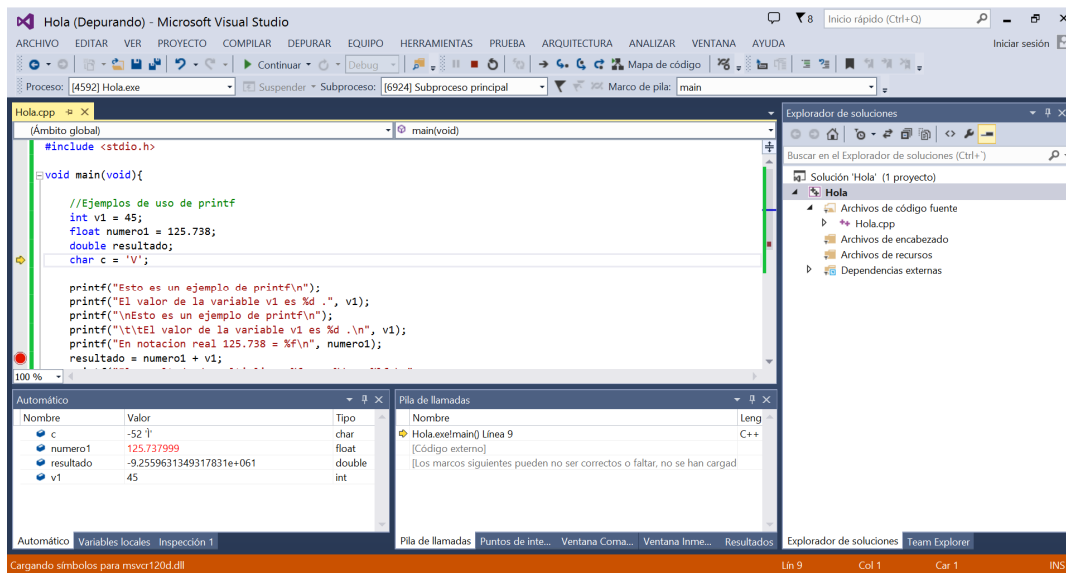
    printf("Esto es un ejemplo de printf\n");
    printf("El valor de la variable v1 es %d .",v2);
    printf("\nEsto es un ejemplo de printf\n");
    printf("\t\tEl valor de la variable v1 es %d .\n",v1);
    printf("En notacion real 125.738 =%f\n", numero1);
    resultado=numero1 + v1;
    printf("El resultado de multiplicar %f por %d es %lf.\n",
           numero1,v1,resultado);
    printf("El codigo ASCII de la letra %c es %u.\n",c,c);
}
```

Se pide:

1. Escribir el programa en el entorno de programación.
2. Corregir los posibles errores de traducción.
3. Ejecutar el programa y comprobar que los resultados son correctos (no hay errores de ejecución). Para ello, habrá que seleccionar la opción *Depurar => Iniciar depuración*.
4. Ejecutar el programa paso a paso (instrucción por instrucción), pulsando la tecla F10 para ello, e ir visualizando cómo cambia el valor de las variables introducidas en la ventana de *Inspección*, ventana situada en la parte inferior del monitor, o situándose con el cursor del ratón sobre ellas. Se muestra un ejemplo del aspecto de la pantalla en la página siguiente.

2) Reescribir el programa anterior para que la operación de suma de las variables *numero1* y *v1* se realice mediante una función; el resultado deberá ser devuelto al programa o función principal. **Nota:** en este caso, cuando se ejecuta un programa paso a paso, deberemos utilizar la tecla F11 para seguir la depuración dentro de una función cuando dicha ejecución se posicione en la invocación a la misma.

Imagen de ejemplo del Ejercicio 1, apartado 4:



Además, en este ejemplo puede observarse la existencia de un punto de interrupción o *break point* (marcado mediante un círculo rojo en la zona izquierda del código). Un punto de interrupción nos permite avanzar en la depuración del programa hasta que es alcanzada la instrucción marcada con el mismo. Si durante la depuración del programa pulsamos el botón marcado como Continuar (botón con un triángulo verde), se ejecutarán las instrucciones comprendidas entre el punto actual de depuración, marcado con la flecha amarilla, y el punto de interrupción, marcado, como hemos dicho, mediante un círculo rojo.

2. Librerías estáticas y dinámicas

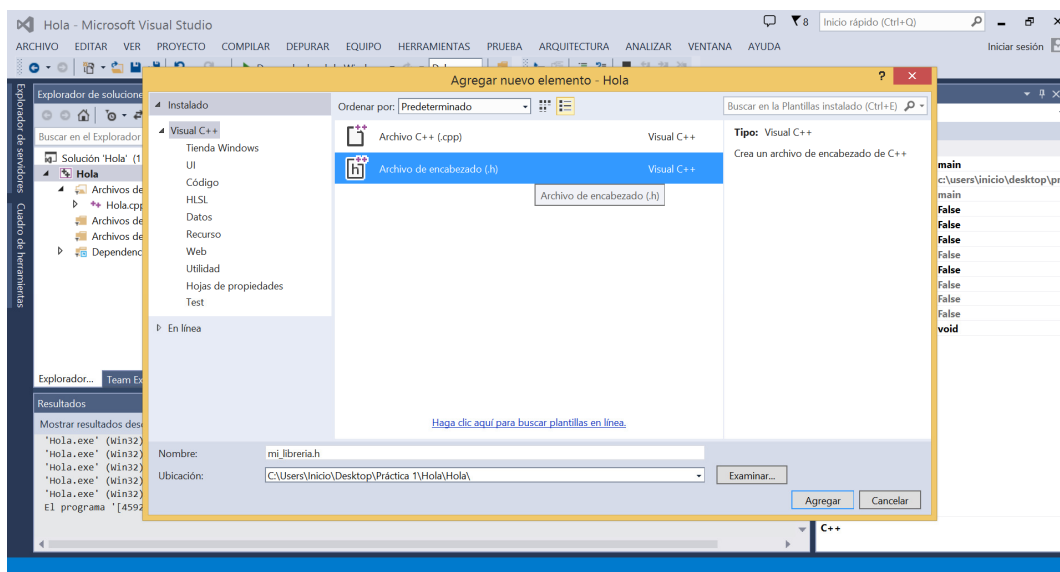
Creación de librerías:

Las librerías son ficheros de código que contienen funciones que solemos usar frecuentemente y que, por lo general, están agrupadas de acuerdo a su utilidad. Por ejemplo, la librería `stdio.h` tiene funciones para controlar la entrada y salida de datos: `printf()`, `scanf()`, etc. En esta segunda parte de la práctica aprenderemos a crear librerías estáticas y dinámicas para, posteriormente, escribir un programa en el cual usaremos dichas librerías.

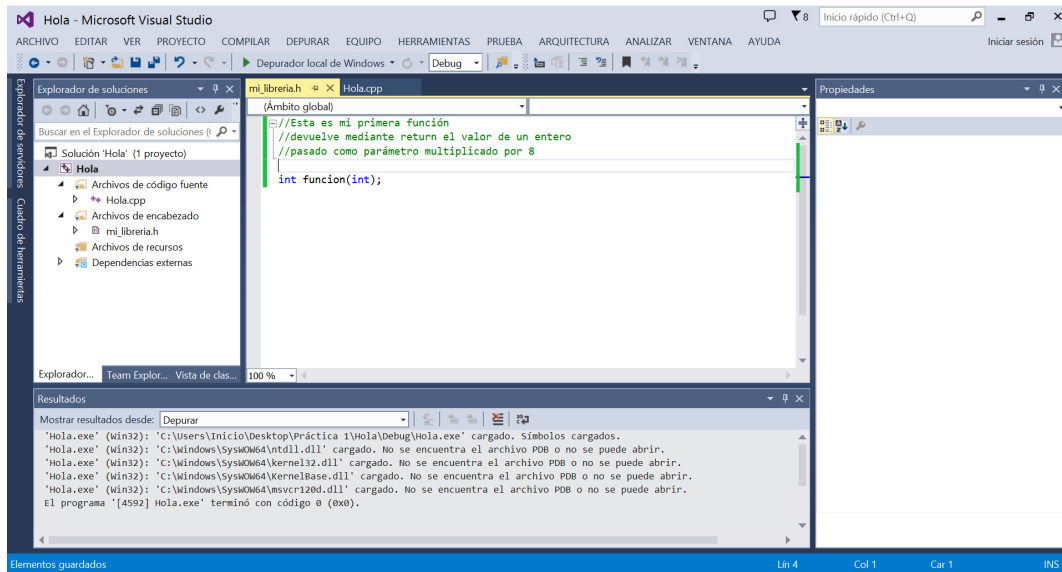
Las librerías, en general, están formadas por un fichero de cabecera (con extensión `.h`) y un fichero de cuerpo o código (con extensión `cpp`). Sin embargo, existen diferencias que permiten diferenciar entre librerías estáticas y dinámicas. Comenzaremos viendo cómo podemos crear una librería estática.

Los pasos a seguir para crear una librería estática de ejemplo serán los siguientes:

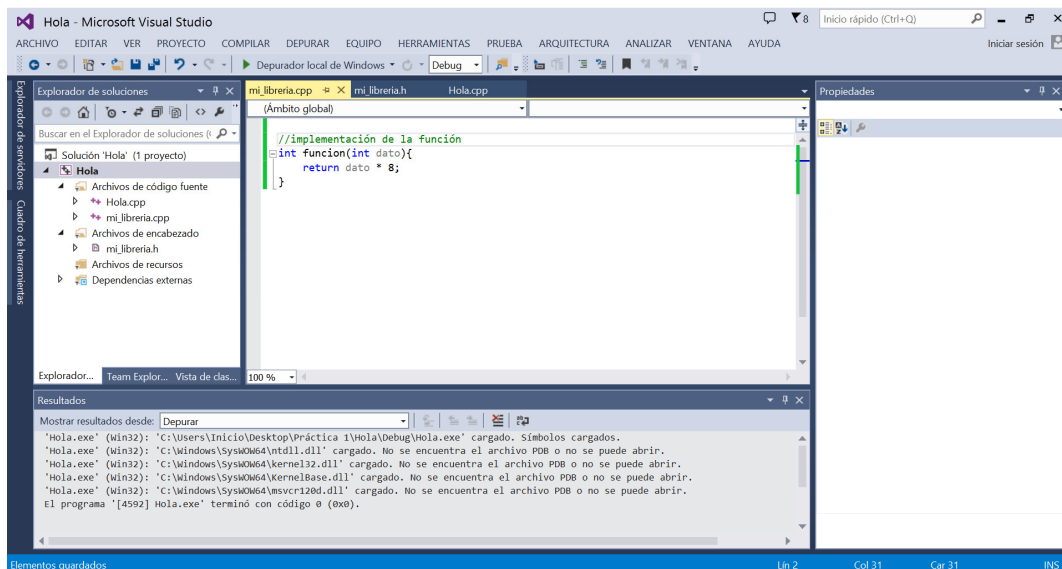
- 1) Crearemos un proyecto y un fichero donde guardar el código C++ de nuestro programa principal, de la misma forma como hemos hecho hasta ahora para la primera parte de esta práctica. Este fichero nos permitirá probar el uso de la librería que vamos a desarrollar.
- 2) Ahora agregamos un nuevo fichero, la cabecera de la librería. En este fichero .h que vamos a crear, únicamente se escribirán los prototipos de las funciones que queramos escribir en nuestra librería. Para esto, seleccionaremos Proyecto => Agregar nuevo elemento => Archivo de encabezado. Debemos dar un nombre al fichero cabecera de la librería, el nombre será `mi_libreria.h`.



Una vez hecho esto veremos que dicho fichero aparece ubicado dentro de la carpeta de Archivos de encabezado, en el Explorador de soluciones, y podremos escribir el prototipo de nuestra función, como se muestra en la siguiente imagen.



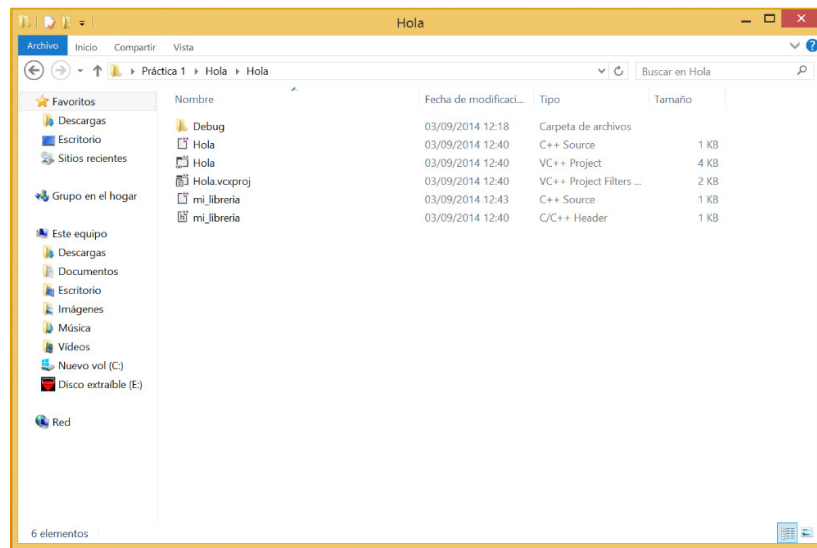
3) A continuación, escribiremos la implementación de la función *funcion()* anterior, dentro del fichero de cuerpo de la librería. Para ello, vamos nuevamente a Proyecto => Agregar nuevo elemento => Archivo de C++ (.cpp). Deberemos escribir el mismo nombre que hemos empleado para el fichero de cabecera, es decir, mi_libreria.cpp, pues obligatoriamente deben coincidir los nombres de ambos ficheros (cabecera y cuerpo). Una vez hecho esto, el fichero *mi_libreria.cpp* aparecerá en la carpeta de Archivos de código fuente, dentro del Explorador de soluciones.



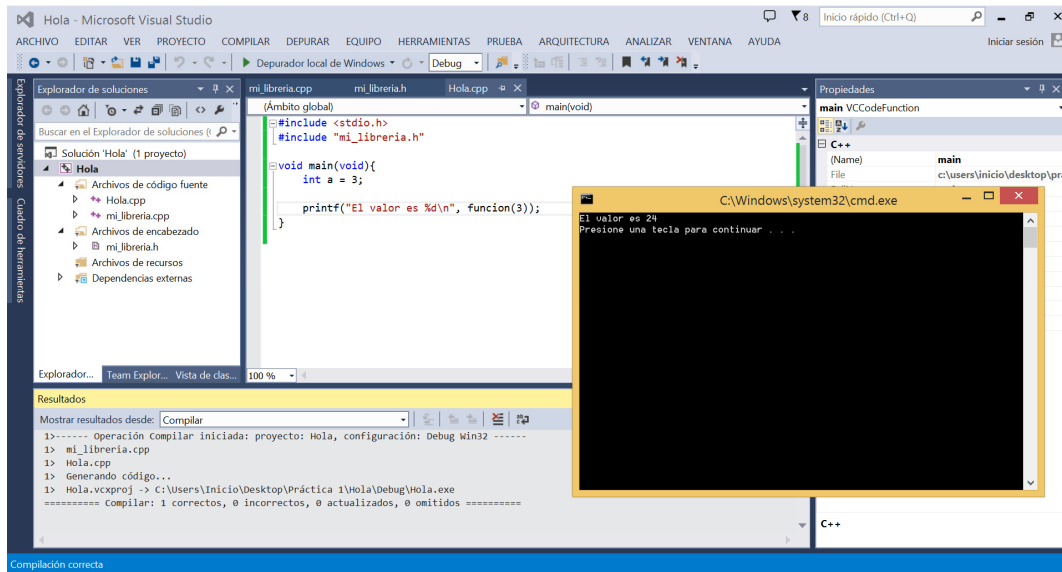
En este fichero escribiremos la implementación de las funciones que queramos detallar en la librería. Pero además, podríamos detallar funciones privadas, es decir, funciones que solamente podrían ser usadas desde el cuerpo de la librería, al no haber sido publicadas en el

fichero de extensión .h. Es decir, la cabecera especifica cuáles, de entre todas las funciones implementadas dentro del cuerpo de la librería, pueden ser empleadas por el programa que haga uso de dicha librería.

Con estos pasos ya tenemos creada nuestra librería estática. Si vamos al Explorador de Windows, a nuestra carpeta de trabajo etiquetada en el escritorio como *Práctica 1*, podremos ver que se han creado algunos ficheros dentro de la misma. De entre todos, los ficheros de la librería son: *mi_libreria.h* y *mi_libreria.cpp*. En el futuro, y si queremos utilizar nuestra recién creada librería, sólo serán necesarios los 2 ficheros anteriores.



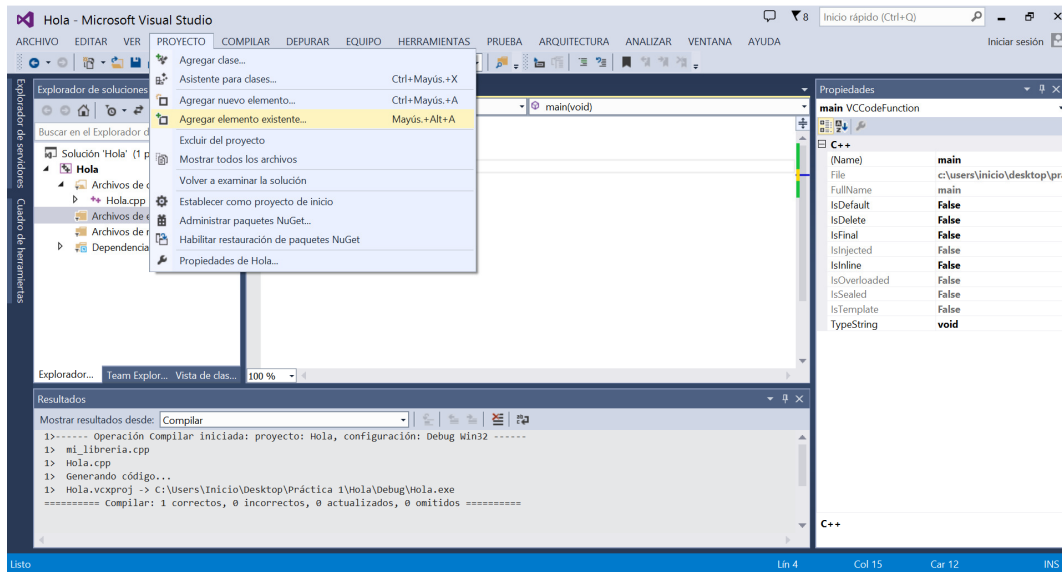
Para terminar, y con el fin de poder emplear la librería en nuestro programa utilizaremos la sentencia `#include` para añadirla al programa, solamente con un diferencia, emplearemos comillas dobles en lugar de los símbolos `<>`.



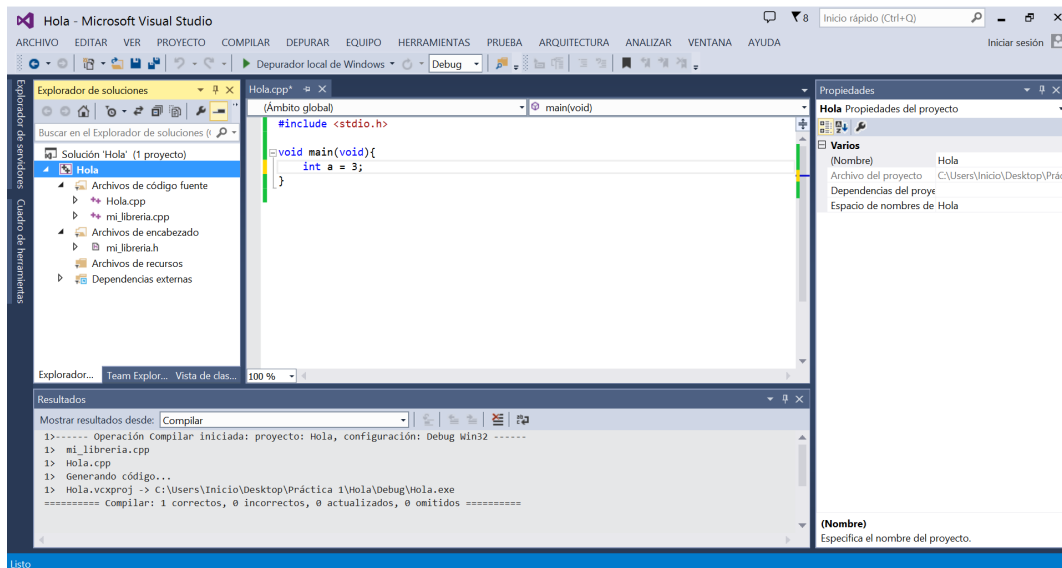
Uso de librerías estáticas:

A continuación, comentaremos cómo añadir una librería estática a un proyecto en general, para poder ser utilizada dentro del mismo; ya sea ésta una librería que hayamos creado anteriormente, o suministrada externamente (desarrollada por un fabricante, por ejemplo). Los pasos a seguir para añadirla a nuestro proyecto serán los siguientes:

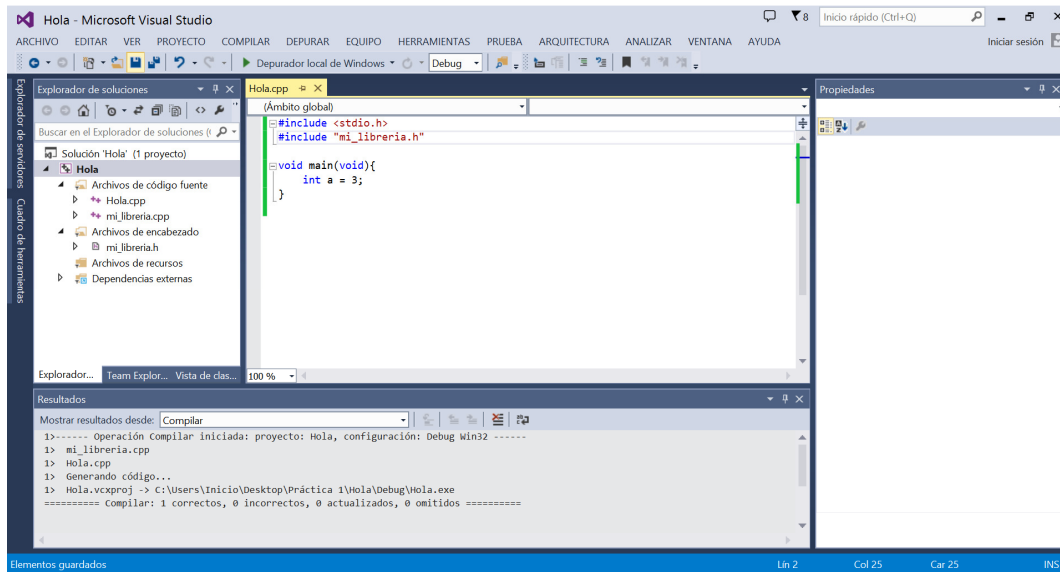
- 1) Crearemos un proyecto y un fichero donde guardar el código C++ del programa, como hemos hecho hasta ahora para las prácticas.
- 2) Abriremos la carpeta que tendremos en el escritorio (o en la ubicación en la que se encuentre almacenado el proyecto) y localizaremos el fichero con extensión .cpp que hemos creado para ubicar el código de nuestro programa principal. En dicho directorio copiaremos los ficheros de mi_libreria.h y mi_libreria.cpp.
- 3) Ahora, agregaremos un fichero de cabecera existente (concretamente el que hemos copiado en el paso anterior). Para esto, seleccionaremos Proyecto => Agregar elemento existente, seleccionando el archivo .h. El programa detectará que se trata de un fichero de cabecera y lo ubicará correctamente en la carpeta de Archivos de encabezado, dentro del Explorador de soluciones.



4) Repetiremos los pasos dados anteriormente, solo que ahora agregaremos el fichero de cuerpo de la librería, con extensión .cpp.



Recuerda que, para poder emplear la librería estática deberemos utilizar la sentencia *#include*, con el nombre de la librería entre comillas, tal y como se muestra en la siguiente imagen.



Ejemplo de uso de librerías:

Una vez visto cómo crear y utilizar librerías estáticas, descargaremos los ficheros *mi_led.h* y *mi_led.cpp* del espacio virtual de la asignatura, pues trabajaremos con ellos. Una vez descargados, deberás incluirlos en el proyecto que se realizará para resolver esta práctica, de la forma que hemos comentado previamente.

Esta librería contiene una función que permite encender o apagar el led del teclado que se le especifica como parámetro. Aunque no veremos cómo opera esta función, puedes consultar los ficheros de cabecera y de cuerpo para ver cómo se encuentra implementada la librería, así como para ver qué valores son los que te permiten controlar el encendido/apagado de un led en concreto. **Nota.** Los ficheros de cuerpo de librería también pueden y deben tener sentencias *#include*, si fuera necesario utilizar tipos de datos o funciones disponibles, a su vez, en otras librerías. En este caso, el fichero *mi_led.cpp* incluye la librería *Windows.h*, pues le es necesaria.

Ejercicios:

1) Dada una matriz 4x4 de números enteros declarada en la función principal, se pide realizar un programa que permita, mediante opciones, acceder a las distintas funciones que a continuación se detallan:

- Escribir una función *lectura()*, que permita al usuario introducir los valores a almacenar en la matriz.

- b) Escribir una función `escribir()`, que permita al usuario modificar el valor de una posición en concreto de la matriz. La matriz, la posición a modificar y el nuevo valor serán pasados a la función mediante parámetros.
- c) Escribir una función `imprimir()`, que permita imprimir los valores almacenados en la matriz.
- d) Escribir una función `borrar()` que nos permita borrar todos los valores almacenados en la matrices. Nota. Esta función deberá escribir en todas las posiciones de la matriz el valor 0.

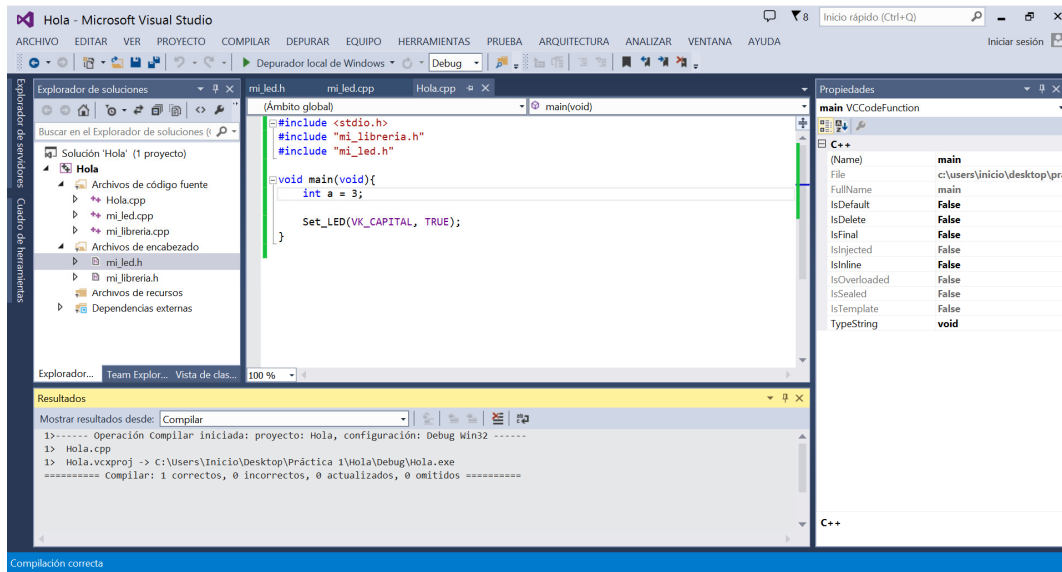
Además, deberán implementarse, mediante una librería estática, las siguientes funciones:

- e) Escribir una función `multiplica()` que realice la multiplicación de una matriz por un valor entero.
- f) Escribir una función `traspuesta()` que obtenga la matriz traspuesta de la matriz que se le suministra como parámetro, almacenando el resultado en la matriz original pasada como parámetro. Es decir, se modifica la matriz original.

2) Tomando como punto de partida el programa realizado en el ejercicio anterior, emplear la librería estática `mi_led` para:

- a) Encender el primero de los leds si todos los valores de la matriz son cero.
- b) Encender el led central del teclado si existe en la matriz al menos un valor mayor que 100.
- c) Encender el último de los leds del teclado si existe en la matriz algún valor negativo.

Estas operaciones de comprobación se realizarán cuando el usuario del programa seleccione la opción de imprimir matriz; y el valor de los leds se actualizará cada vez que el usuario seleccione nuevamente la opción de imprimir. A continuación, puede verse un ejemplo de invocación a la función `Set_LED()`, incluida dentro de la librería `mi_led`.



3) Busca información en relación a las instrucciones siguientes que aparecen en el fichero de cabecera de la librería `mi_led`:

```
#ifndef __MI_LED_H_INCLUDED__
#define __MI_LED_H_INCLUDED__

.....

#endif
```

- ¿Para qué se utilizan estas instrucciones? ¿Cuál es su función?
- Añade dichas instrucciones en la librería creada en el ejercicio 2.

Como resumen a lo que hemos visto, en relación a las librerías estáticas, podríamos decir que el código de las mismas se copia en nuestro programa de manera automática cuando lo compilamos. Es decir, una vez que tenemos el ejecutable de nuestro programa (el archivo con extensión `exe`), podríamos borrar los archivos y nuestro programa seguiría funcionando correctamente; ya que tiene una copia de todo aquello que necesita (**nota**. sólo se copiará aquella parte de la librería que sea necesaria; por ejemplo, si la librería tiene una serie de funciones, sólo se copiarán las que sean utilizadas).

Librerías dinámicas:

Finalmente, y para terminar esta primera práctica, veremos cómo emplear librerías dinámicas en C (conocidas como DLLs), y cuáles son las diferencias que presentan frente a las librerías estáticas.

A diferencia de una librería estática, una librería dinámica no se copia en nuestro programa al compilarlo, es decir, cuando tengamos el archivo ejecutable del programa, y cada vez que éste necesite algo (una función) de la librería, irá a buscarlo a ella. Si borramos la librería (el archivo dll de la misma), nuestro programa dará un error de ejecución.

Debido a que la creación de una librería dinámica incluye el dominio de una serie de instrucciones específicas destinadas a tal fin, la explicación de su programación requiere algo más de detalle; es necesario añadir instrucciones específicas de C y C++ para poder hacer uso de la misma (principalmente, de cara a la comunicación con nuestro programa principal). Por ello, no veremos en esta práctica la creación de una librería dinámica, pues lo que realmente nos interesa es el uso de la misma en nuestros programas, independientemente de la sintaxis que sea necesaria para programar las funciones contenidas en ella. Con todo, el alumno puede consultar la ayuda MSDN de Microsoft, con el fin de obtener más información al respecto (<http://msdn.microsoft.com/es-es/library/ms235636%28v=vs.120%29.aspx>).

En el siguiente punto, emplearemos una librería dinámica para realizar funciones matemáticas básicas entre dos variables de tipo real.

Ejemplo de uso de librerías dinámicas:

Descargaremos los ficheros *Math_v2.h*, *Math_v2.lib* y *Math_v2.dll* del espacio virtual de la asignatura, pues trabajaremos con ellos. Una vez descargados, deberás incluirlos en un proyecto vacío que se realizará para resolver esta parte de la práctica (nuevamente, Aplicación de consola y Proyecto vacío), de la forma que indicaremos a continuación. La función de cada uno de estos ficheros es la siguiente:

- *Math_v2.h*. Fichero de cabecera de la librería, que contiene las funciones que podemos emplear, es decir, las funciones que han sido implementadas en la librería. Este fichero nos permite saber cuáles son las funciones accesibles/disponibles en la librería.
- *Math_v2.lib*. Fichero necesario para compilar el programa que hace uso de la librería.
- *Math_v2.dll*. Fichero necesario para la ejecución del programa que hace uso de la librería.

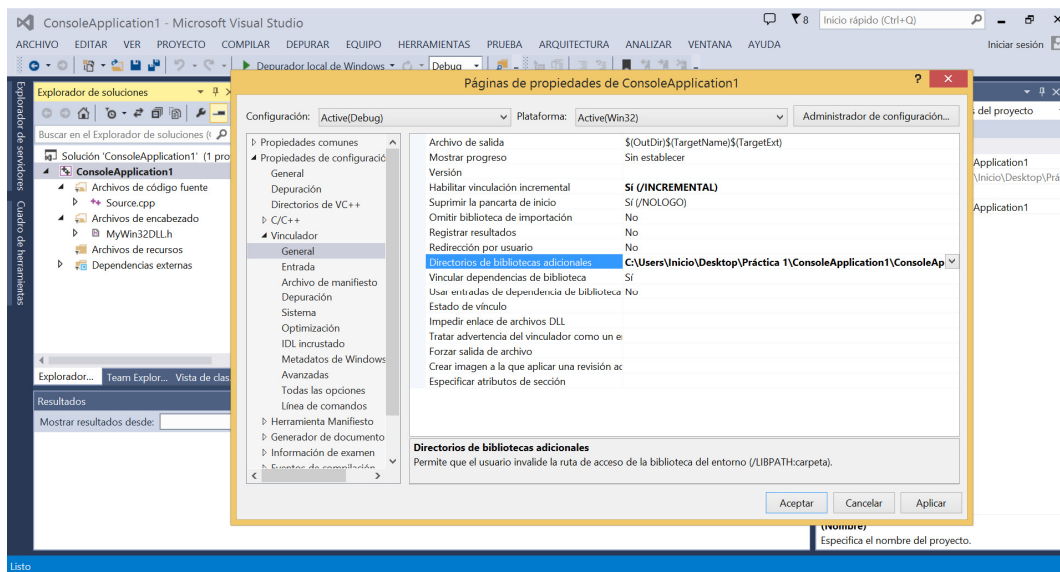
Como se comentó previamente, esta librería contiene las funciones de *suma*, *resta*, *multiplicación* y *división* de dos números reales (de tipo *double*), devolviendo el resultado mediante *return* (también un valor de tipo *double*).

Una vez tengamos listo el nuevo proyecto, realizaremos los siguientes pasos para incluir la librería dinámica.

1) Lo primero de todo, localizaremos la carpeta del nuevo proyecto en la que se encuentra el archivo principal (con extensión .cpp), y procederemos a copiar los tres archivos de la librería dinámica en dicho directorio; igual que realizamos para las librerías estáticas.

2) De la misma forma que agregamos la cabecera de una librería estática (fichero con extensión .h), deberemos agregar al proyecto el fichero *Math_v2.h*, la cabecera de la librería dinámica (ver punto 3 en la página 13).

3) A continuación, configuraremos el proyecto para que pueda localizar el fichero *Math_v2.lib*, fichero necesario para poder compilar el programa anterior. Para ello, abriremos las propiedades del proyecto creado con el botón derecho del ratón sobre su nombre, en el Explorador de soluciones. Una vez hecho esto, localizaremos la entrada Vinculador y dentro de esta, el campo Directorio de bibliotecas adicionales, dentro de la categoría General. Es ahí donde añadiremos la ruta (carpeta) que contiene el citado fichero; que deberá estar en la carpeta principal del proyecto (detallado en el punto 1).



4) Y dentro del mismo apartado de propiedades del proyecto, también deberemos añadir explícitamente el fichero *Math_v2.lib* en el campo Dependencias adicionales del apartado Entrada, dentro de las opciones del Vinculador.

Llegados a este punto, el siguiente código muestra un ejemplo de uso de la librería dinámica en nuestro programa:

```
#include <stdio.h>
#include "Math_v2.h"

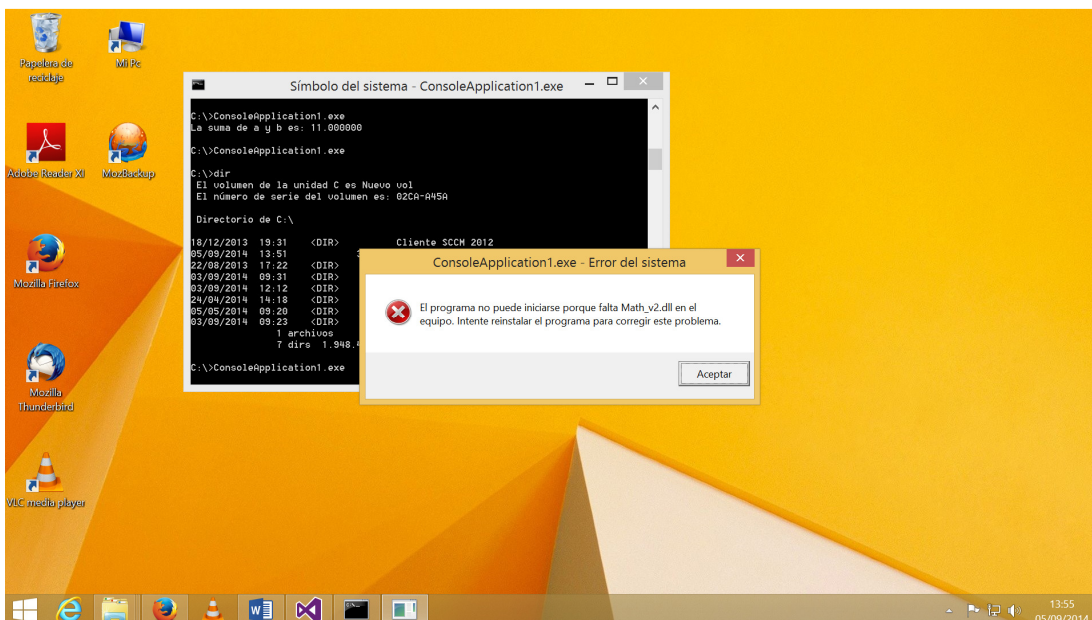
void main(void){
    int a=5,b=6;

    //declaración de variable de tipo librería
    //class math en el fichero Math_v2.h
    math lib;

    printf("La suma de a y b es: %f\n",lib.suma(a,b));
}
```

Prueba a escribir dicho código, compila el programa y trata de ejecutarlo.

4) El fichero *Math_v2.dll*, es el fichero que necesitaremos copiar junto al ejecutable (archivo de extensión .exe) del programa realizado, cuando queramos ejecutar el programa en otro equipo o desde una ubicación diferente; si no es así el sistema dará un error, al no encontrar la librería. Puedes comprobar el error que el sistema muestra cuando no encuentra la librería, mostrado en la siguiente pantalla:



Para ello, copia solamente el fichero ejecutable de tu aplicación en el directorio raíz del equipo, abre una consola de comandos y ejecuta el programa. Una vez visto el error, prueba ahora a copiar, además, el fichero *Math_v2.dll*.

Ejercicios:

- 1) Realiza un programa que, empleando todas las funciones proporcionadas por la librería *Math_v2*, sume, reste, multiplique y divida dos variables reales leídas por teclado y solicitadas al usuario.
- 2) ¿Qué sucede si el valor de b es cero? ¿Y si a y b son cero? Comprueba la respuesta que el programa ofrece al emplear la librería. Realiza un programa que divida por cero y compruebe la respuesta que se obtiene.
- 3) ¿Podemos realizar una función suma dentro de nuestro programa principal? Escribe, dentro del mismo proyecto realizado para esta práctica, una función que sume dos números enteros y devuelva mediante `return` el resultado. **Nota.** Este ejercicio está propuesto con el fin de comprender el porqué de la declaración de una variable de tipo librería para hacer uso de la función suma de la *dll*.

Ventajas e inconvenientes. Librerías estáticas vs. Librerías dinámicas:

En general, podemos decir que:

1. Un programa que haga uso de librerías estáticas es más grande; se hace copia de todo lo que necesita.
2. Un programa que haga uso de librerías estáticas se puede llevar a otro ordenador (PC) sin necesidad de incluir las librerías. Solamente necesitamos el fichero `.exe`.
3. La utilización de librerías dinámicas presenta la ventaja de poder ser corregidas sin necesidad de compilar nuevamente el programa. Por ejemplo, si queremos cambiar la librería para corregir un error o cambiar el comportamiento de la misma.