

SISTEMAS INFORMÁTICOS

GITT, GIT, GIST Y GIEC

PRÁCTICA 1:

Entorno de desarrollo y construcción de un programa en C

UAH, Departamento de Automática, ATC-SOL

<http://atc1.aut.uah.es>

OBJETIVOS

- Conocer el entorno de desarrollo Microsoft Visual C++
- Editar, compilar y ejecutar un programa sencillo
- Aprender a depurar un programa

TEMPORIZACIÓN

- Inicio de la práctica: Semana del 7 de Septiembre
- Tiempo de desarrollo de la práctica: 2 semanas. Finalizarla antes del examen parcial 1
- Evaluación: ver fechas en <http://atc1.aut.uah.es>.

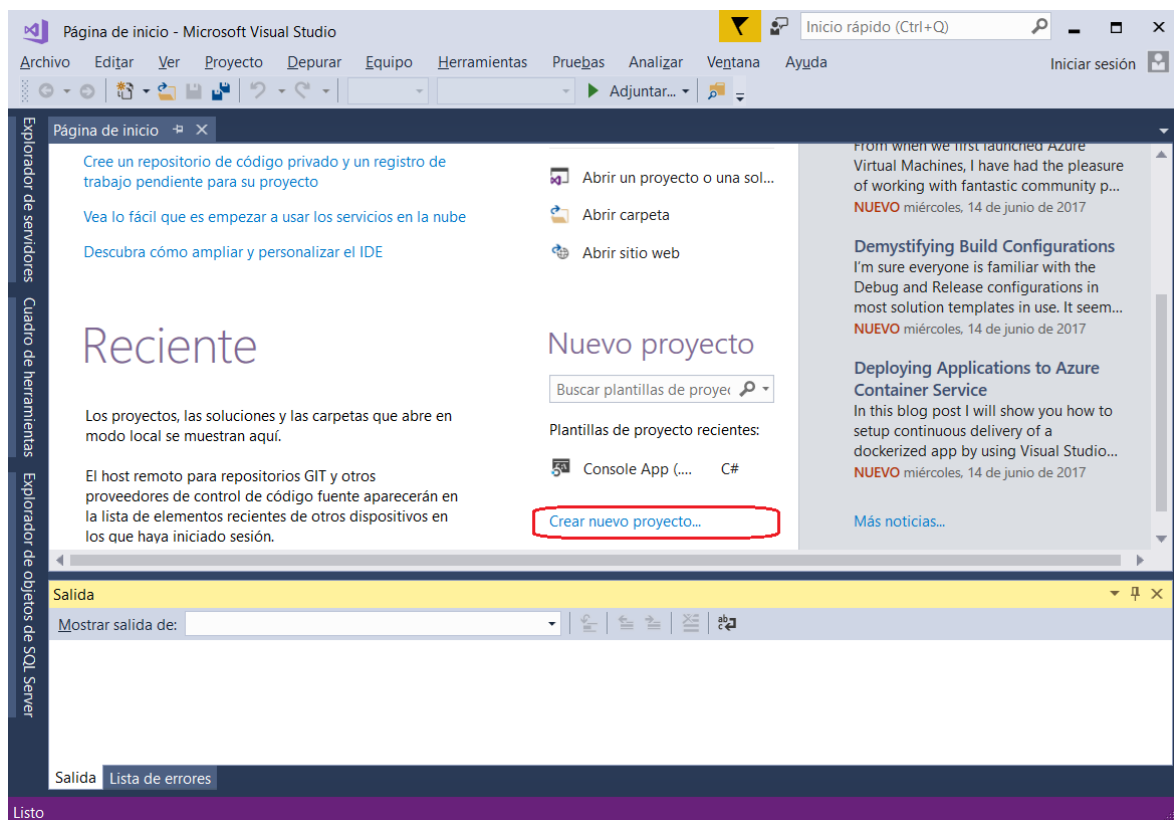
1. Introducción a Microsoft Visual C++

Durante el curso emplearemos el entorno de desarrollo de *MS Visual Studio Community*, que es la versión que está instalada en los laboratorios.

Para cada uno de los programas que se realicen se creará un proyecto de trabajo.

MS Visual Studio nos permite crear dicho proyecto, así como editar, compilar, enlazar, depurar y ejecutar nuestro programa. Además, provee facilidades para gestionar proyectos compuestos por varios archivos fuente, bibliotecas, etc.

Para iniciar el programa, hacer doble clic en el icono del escritorio: *MS Visual Studio*. Aparecerá la siguiente ventana:



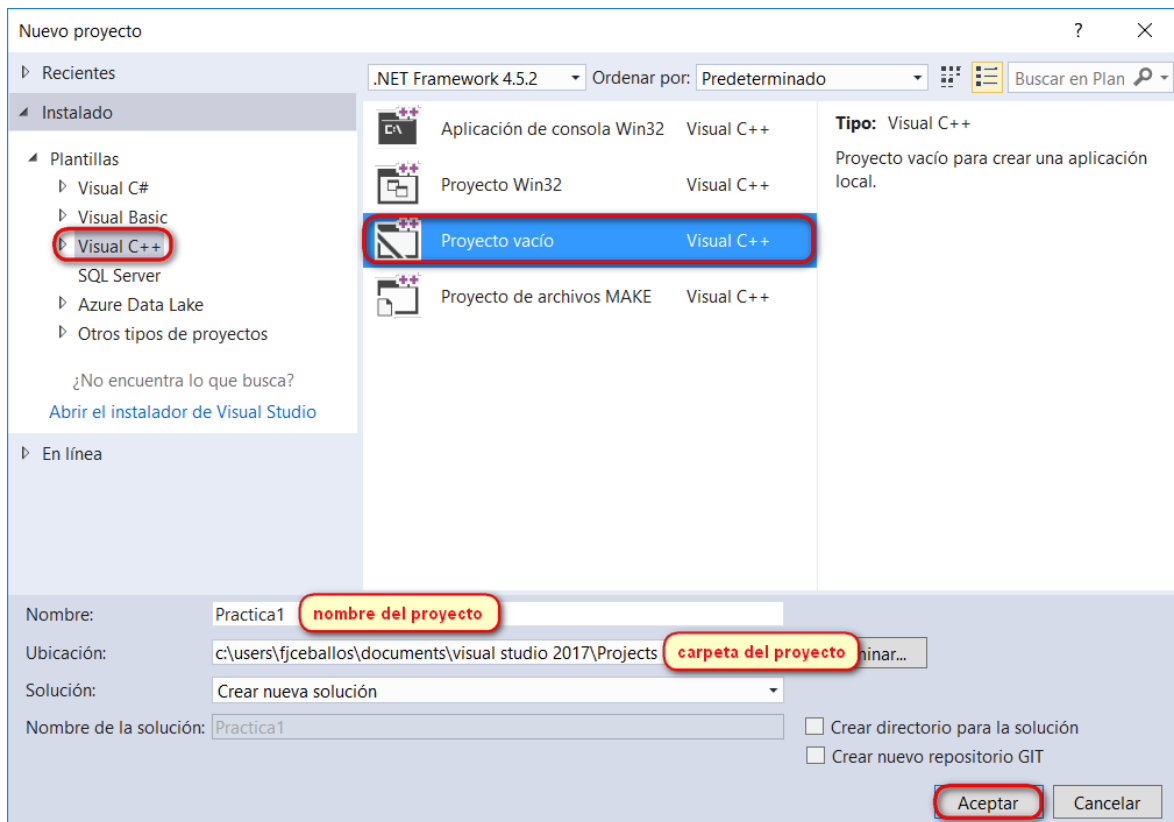
Para poder escribir un programa y crear un fichero ejecutable debemos seguir los siguientes pasos:

1. Crear un proyecto de tipo aplicación de consola.
2. Añadir un fichero al proyecto. Esto nos permitirá escribir nuestro programa.
3. Compilar el programa. Nos permite conocer los errores cometidos al escribir el programa para poder así corregirlos.
4. Enlazar y ejecutar el programa.

En la zona inferior de la pantalla aparece una ventana con la pestaña *Lista de errores* y otra con la de *Salida* (observe las pestañas en el fondo de la ventana) En esta ventana se van a mostrar los mensajes referentes a compilación, enlace y ejecución del programa

2. Crear un proyecto.

En la ventana de la figura anterior hacer clic en *Crear nuevo proyecto...* o bien, a partir de la barra de menús selecciones *Archivo > Nuevo > Proyecto*. Nos Aparecerá una ventana como la siguiente:

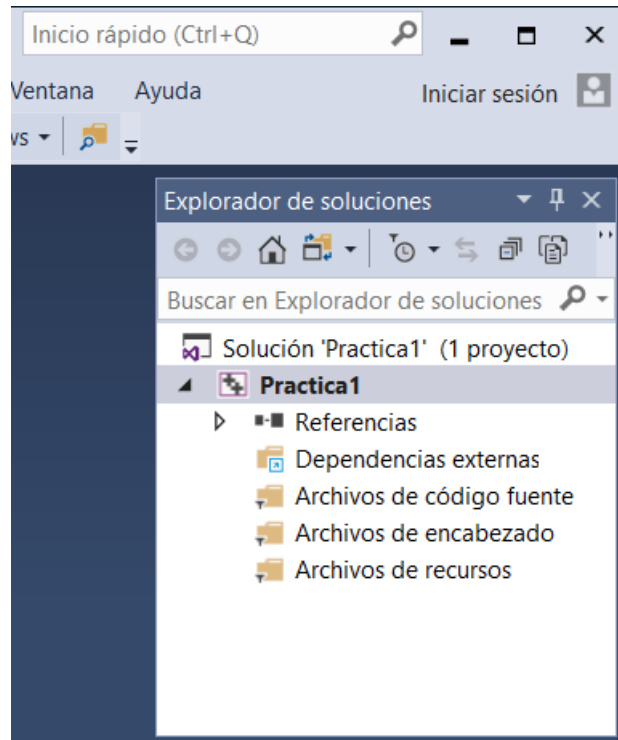


Observe que el tipo de proyecto de Visual C++ que debe seleccionar es “Proyecto vacío”. En la parte inferior debemos escribir el nombre que queremos darle a nuestro proyecto, por ejemplo, *Practica1*, y el directorio donde queremos guardarlo, por ejemplo *C:\...\visual studio 2017\Projects* (asegurándonos previamente que dicho directorio existe en el disco duro). La opción *Crear directorio para la solución* no tiene sentido que esté habilitada para una solución con un solo proyecto.

Después de pulsar el botón “Aceptar” se creará el directorio *C:\...\visual studio 2017\Projects\Practica1* donde están los ficheros propios del proyecto, entre los que destacamos:

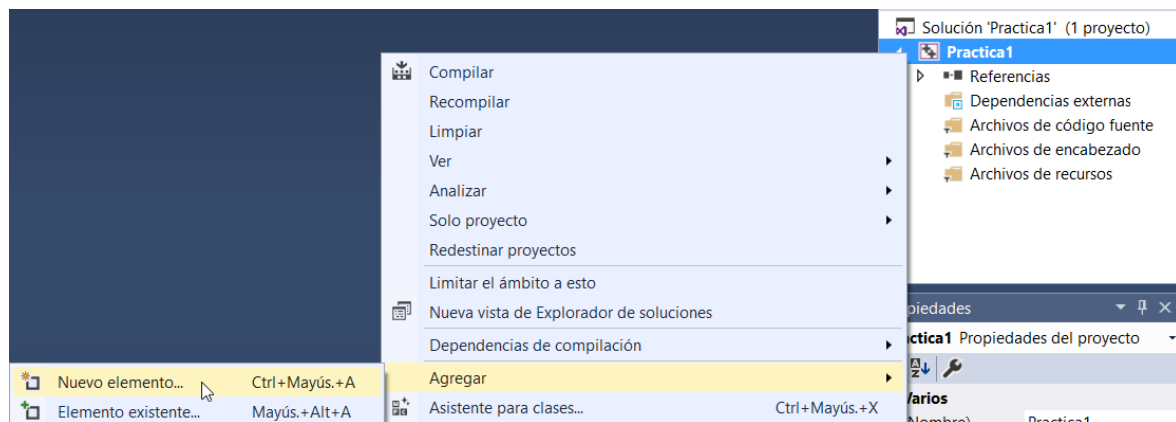
- *Practica1.vcxproj*: almacena la información específica para este proyecto: ficheros de código fuente, binarios, etc. que lo componen y las órdenes que se deben ejecutar para generar el fichero ejecutable.
- *Practica1.sln*: es el archivo de solución. Una solución puede administrar uno o más proyectos.

Cómo puede observar **no** hay ningún fichero fuente “*.c” ó “*.cpp” en el proyecto. Ahora es el momento de añadirlos.

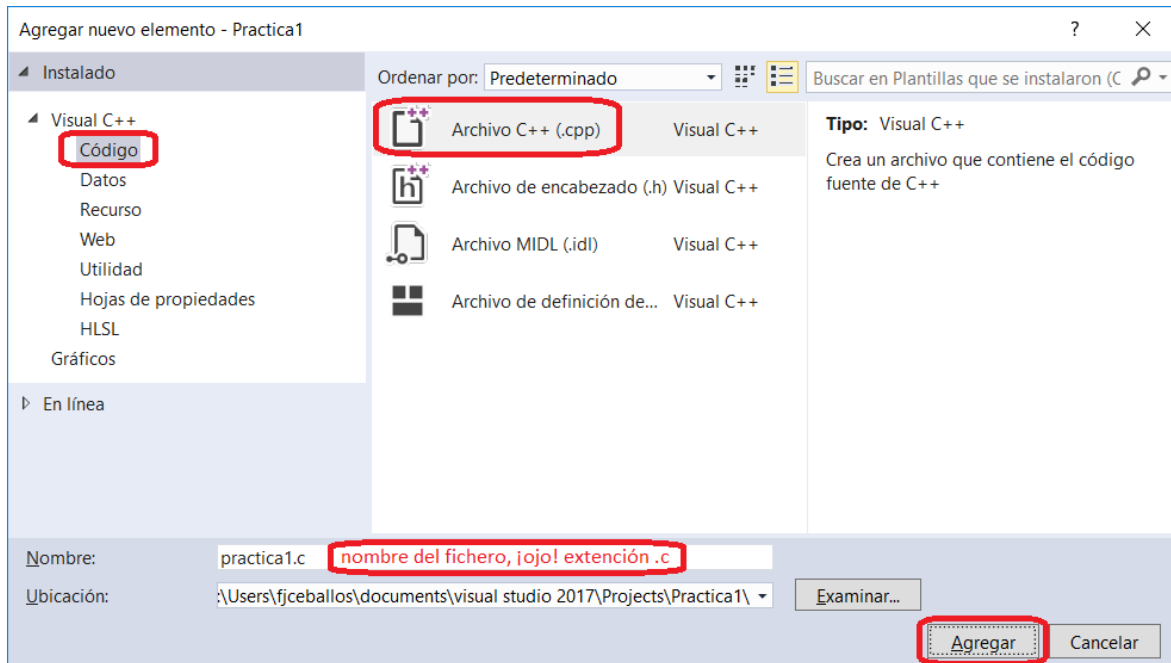


3. Añadir ficheros al proyecto

Haga clic con el botón secundario del ratón sobre el nombre del proyecto y seleccione *Agregar > Nuevo elemento* (puede hacerlo también a través del menú *Proyecto*):



Se mostrará la ventana siguiente:

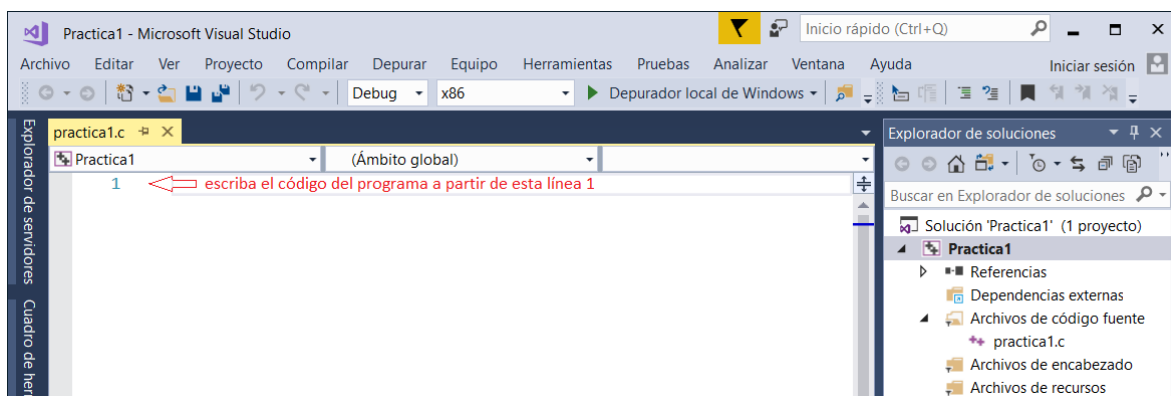


En la cual deben ser seleccionadas las siguientes opciones:

- Visual C++ > Código > Archivo C++ (.cpp)
- Nombre: se refiere al nombre del fichero de código fuente. En este ejemplo hemos llamado al fichero de código que vamos a añadir *practica1.c*; añada la extensión *.c* para evitar que Visual C++ añada la extensión *.cpp* por defecto (esta extensión es para programación orientada a objetos con C++, que no es nuestro caso). El nombre del fichero puede ser cualquiera, pero la extensión tiene que ser *.c*.
- Ubicación: la misma que el proyecto; por ejemplo, *C:\...\visual studio 2017\Projects\Practica1*

Finalmente, haga clic en el botón *Agregar*.

Una vez completado el proceso anterior, nos aparece, tal y como podemos observar en la figura siguiente, una ventana de edición donde podemos escribir nuestro programa en lenguaje C. Podemos observar también en la parte izquierda de la ventana, en el panel *Explorador de soluciones*, que el fichero *practica1.c* se ha incluido en nuestro proyecto en la carpeta *Archivos de código fuente*. El fichero será grabado automáticamente en el disco duro en el directorio *C:\...\visual studio 2017\Projects\Practica1* con el nombre *practica1.c*.



A continuación, vamos a escribir en la ventana de edición un programa para practicar lo que tendremos que hacer en las siguientes prácticas:

```
#define _CRT_SECURE_NO_DEPRECATE
#include <stdio.h>
#include <stdlib.h>

#define SEG_POR_MINUTO 60
#define SEG_POR_HORA 3600
#define SEG_POR_DIA 86400

int main()
{
    int seg;
    float min, horas, dias;

    printf("Este programa le permite pasar de segundos a minutos,\n");
    printf("horas o dias\n\n");
    printf("Escriba cuantos segundos quiere convertir y pulse ENTER: ");
    scanf("%d", &seg);

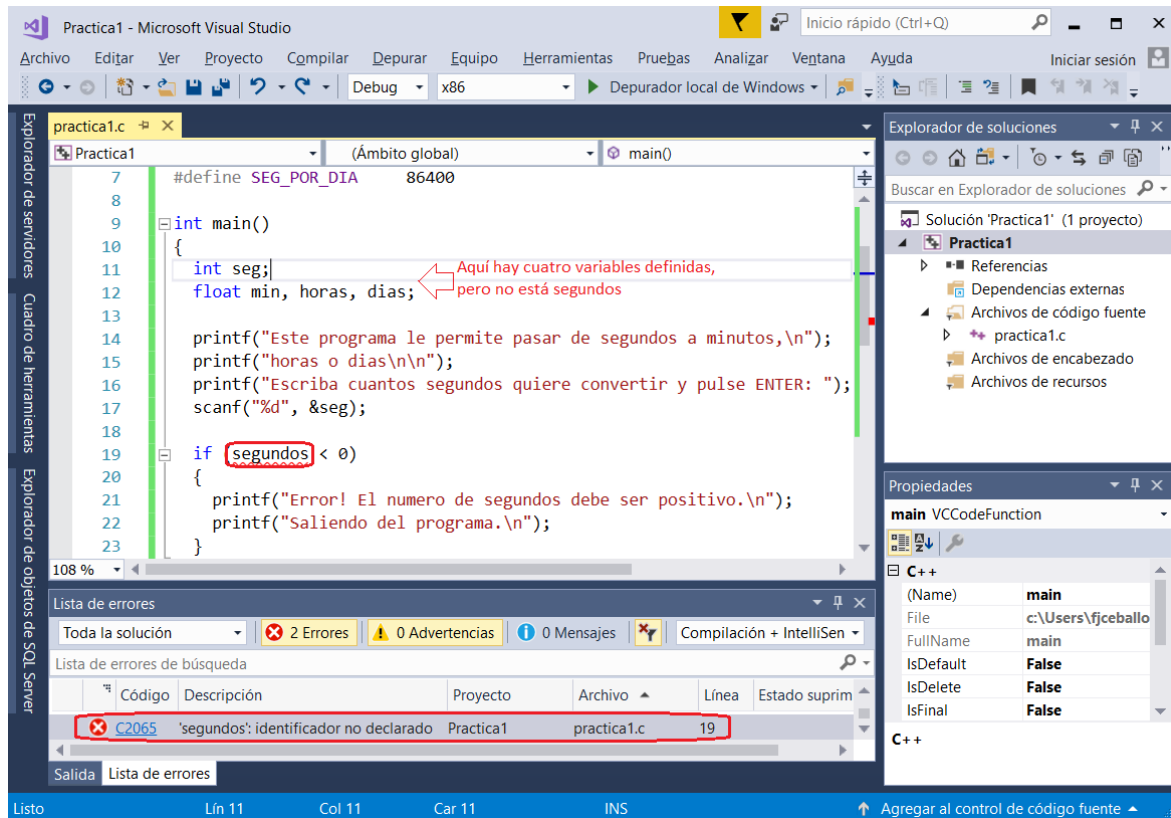
    if (segundos < 0)
    {
        printf("Error! El numero de segundos debe ser positivo.\n");
        printf("Saliendo del programa.\n");
    }
    else
    {
        min = (float)seg / SEG_POR_MINUTO;
        horas = (float)seg / SEG_POR_HORA;
        dias = (float)seg / SEG_POR_DIA;
        printf("%d segundos equivalen a %.3f minutos.\n", seg, min);
        printf("%d segundos equivalen a %.3f horas.\n", seg, horas);
        printf("%d segundos equivalen a %.3f dias.\n", seg, dias);
    }

    system("pause");
    return 0;
}
```

El código de este ejemplo permite calcular los minutos, horas y días a los que equivale un determinado número de segundos que le pedimos al usuario por teclado. No se preocupe ahora si no entiende todos los detalles del mismo. Puede observar que el editor asigna colores a algunas palabras. Observe también que el estilo de edición es importante; un código organizado ayuda a detectar errores y es mucho más fácil de revisar.

Hemos introducido un error en el código, que veremos en el siguiente paso, cuando compilemos el programa.

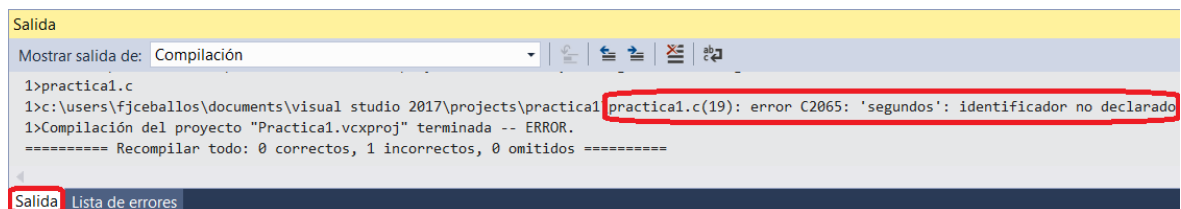
MS Visual Studio está construido para que la mayoría de los errores se visualicen sin necesidad de llegar a compilar el programa, pero no todos los entornos de desarrollo tienen esta característica, por lo que, por esta vez, vamos a obviar esto.



4. Compilar y enlazar.

Compilar significa traducir el código fuente a código ejecutable. Durante este proceso, si alguna parte del código fuente no es traducible porque hemos cometido errores al escribirlo, el compilador nos notificará dichos errores.

Para compilar el programa ejecutamos la orden *Compilar/Recompilar solución* del menú *Compilar*. El resultado de la compilación se nos mostrará en la ventana de *Salida* (debajo de la ventana de edición); en esta ventana veremos los errores cometidos o, en su defecto, un mensaje indicando que no hay errores. A continuación de forma automática se realiza el enlazado y se genera el fichero ejecutable llamado *practical.exe* situándolo en el directorio *C:\...\visual studio 2017\Projects\Practical\Debug*. El resto de archivos que aparecen son generados por *Visual C++* y no debe preocuparse de ellos por ahora.



En nuestro ejemplo hemos introducido un error de compilación para ver la forma en la que se muestra la información del error para proceder a su corrección.

En la ventana de *Salida* podemos observar algunos mensajes, unos calificados como *warning* (aviso) y otros calificados como *error*. Los avisos no detienen el proceso de construir el fichero ejecutable, pero los errores sí. Por eso, estos últimos tenemos que corregirlos y volver a compilar el programa. Observe el *error C2065: 'segundos' identificador no declarado* ocurrido en la línea 19 de *practica1.c*. El identificador *segundos* no se ha declarado; entonces, o la variable no se llama así o, simplemente, se llama así y hay que declararla.

La información relativa a los errores puede verla también en la ventana *Lista de errores*:

Lista de errores						
<div>Toda la solución</div> <div> ✖ 2 Errores ⚠ 0 Advertencias ℹ 0 Mensajes </div> <div> 🔍 <div>Compilación + IntelliSen</div> </div>						
	Códi...	Descripción	Proyecto	Archivo	Línea	Estado suprimido
	E0020	el identificador "segundos" no está definido	Practica1	practica1.c	19	
	C2065	'segundos': identificador no declarado	Practica1	practica1.c	19	

El código de error *E0020* se refiere a un error detectado durante la fase de *Edición* (vea a continuación que la ventana de edición muestra “segundos” subrayado en rojo), y el código *C2065* es un error detectado durante la fase de *Compilación*.

Si hace doble clic sobre la línea que indica el error en la ventana de *Salida* o en la ventana *Lista de errores*, el punto de inserción será situado de forma automática en la ventana de edición en la línea exacta del código fuente donde está el error. Observe que *segundos* aparece subrayado en rojo.

Efectivamente, podemos observar que hemos empleado el identificador *segundos* en lugar del identificador *seg*, probablemente porque nos confundimos.

```

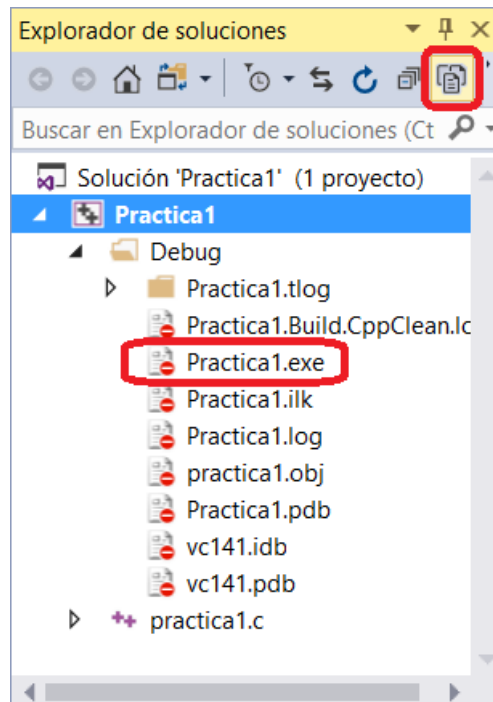
9  int main()
10 {
11     int seg;
12     float min, horas, dias;
13
14     printf("Este programa le permite pasar de segundos a minutos,\n");
15     printf("horas o dias\n\n");
16     printf("Escriba cuantos segundos quiere convertir y pulse ENTER: ");
17     scanf("%d", &seg);
18
19     if segundos < 0)
20     {

```

Corregimos dicho error sustituyendo *segundos* por *seg* y volvemos a intentar la generación, que en esta ocasión se realizará sin errores.

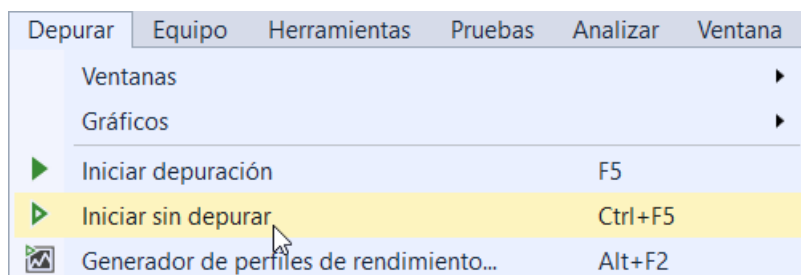
Salida	
Mostrar salida de: <div>Compilación</div>	
1>----- Operación Compilar iniciada: proyecto: Practica1, configuración: Debug Win32 ----- 1>practica1.c 1>Practica1.vcxproj -> c:\users\fjceballos\documents\visual studio 2017\Projects\Practica1\Debug\Practica1.exe 1>Practica1.vcxproj -> c:\users\fjceballos\documents\visual studio 2017\Projects\Practica1\Debug\Practica1.pdb (Partial PDB) ===== Compilar: 1 correctos, 0 incorrectos, 0 actualizados, 0 omitidos =====	
Salida	Lista de errores

De forma automática se ha realizado también el enlazado y se genera el fichero ejecutable llamado *practical.exe* que se guarda en el directorio *C:\...\visual studio 2017\Projects\Practical\Debug*. Puede verlo pulsando el botón *Mostrar todos los archivos* en el *Explorador de soluciones*:

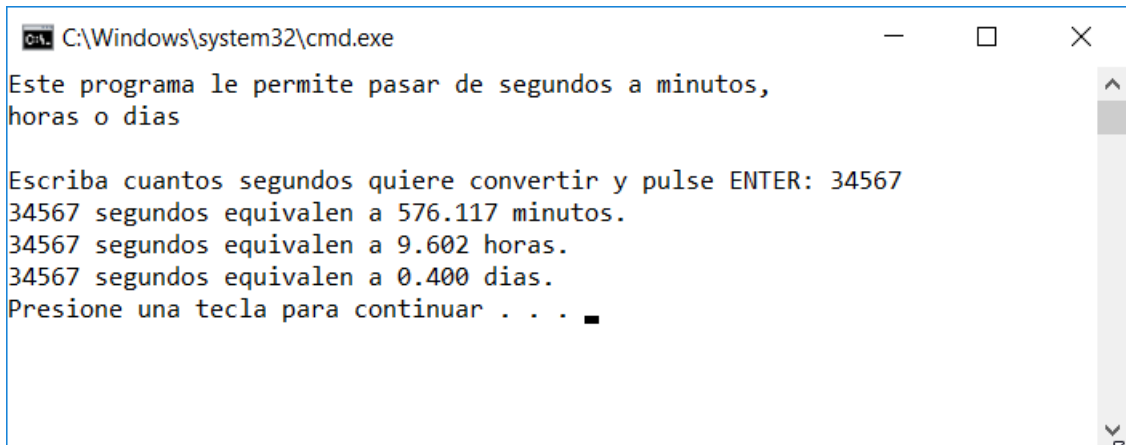


5. Ejecutar el programa

Ya se puede ejecutar el programa. Para ello se debe pulsar *Control+F5* o elegir la orden *Iniciar sin depurar* del menú *Depurar* (si esta orden no está, se puede añadir: *Herramientas > Personalizar*).



El resultado se mostrará en una consola según muestra la figura siguiente. Según esta figura, usted teclea el valor *34567* seguido de la tecla *Entrar* y le será mostrado el resultado:



Como puede comprobarse, el programa funciona correctamente. Pulse cualquier tecla para cerrar la ventana.

6. Depurar el programa.

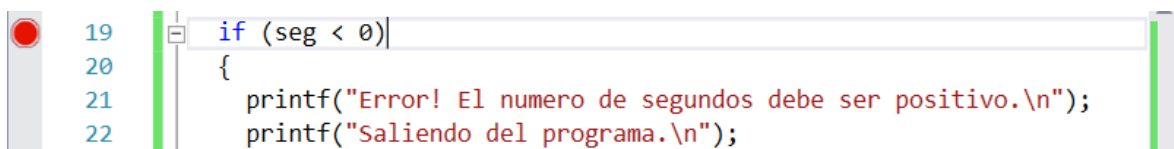
Uno de los aspectos más importantes en el desarrollo de un programa es la depuración, que permite ejecutar el programa de forma controlada, paso a paso, parando su ejecución y visualizando el valor de las variables a medida que avanzamos a lo largo del flujo del programa.

Utilizaremos la depuración cuando un programa, a pesar de carecer de errores de compilación y de haberse ejecutado, no lo hace como nosotros esperábamos. Mediante la depuración podremos ejecutar un programa de forma controlada, es decir, podremos ejecutar paso a paso solamente un grupo de instrucciones y ver el valor que van tomando las variables. Si éste no es el valor que debía tomar la variable en esa parte del programa, indica que el error se ha producido en el código anterior a dicho punto. Por el contrario, si el valor de las variables es el esperado, podemos seguir ejecutando instrucción a instrucción.

Para iniciar la depuración del programa pulsamos F5, o ejecutamos la orden *Iniciar depuración* del menú *Depurar*, o hacemos clic en el botón correspondiente de la barra de herramientas; elija la opción que le sea más cómoda. El programa se ejecutará esperando a que introduzcamos el número de segundos. Si lo hacemos y pulsamos *Entrar*, el programa se ejecutará hasta el final y no habremos conseguido nada; por ello, debemos establecer un punto de parada, que es donde queremos que el programa se detenga para realizar la depuración. Otra opción es iniciar la depuración del programa pulsando F10 o F11 en lugar de F5.

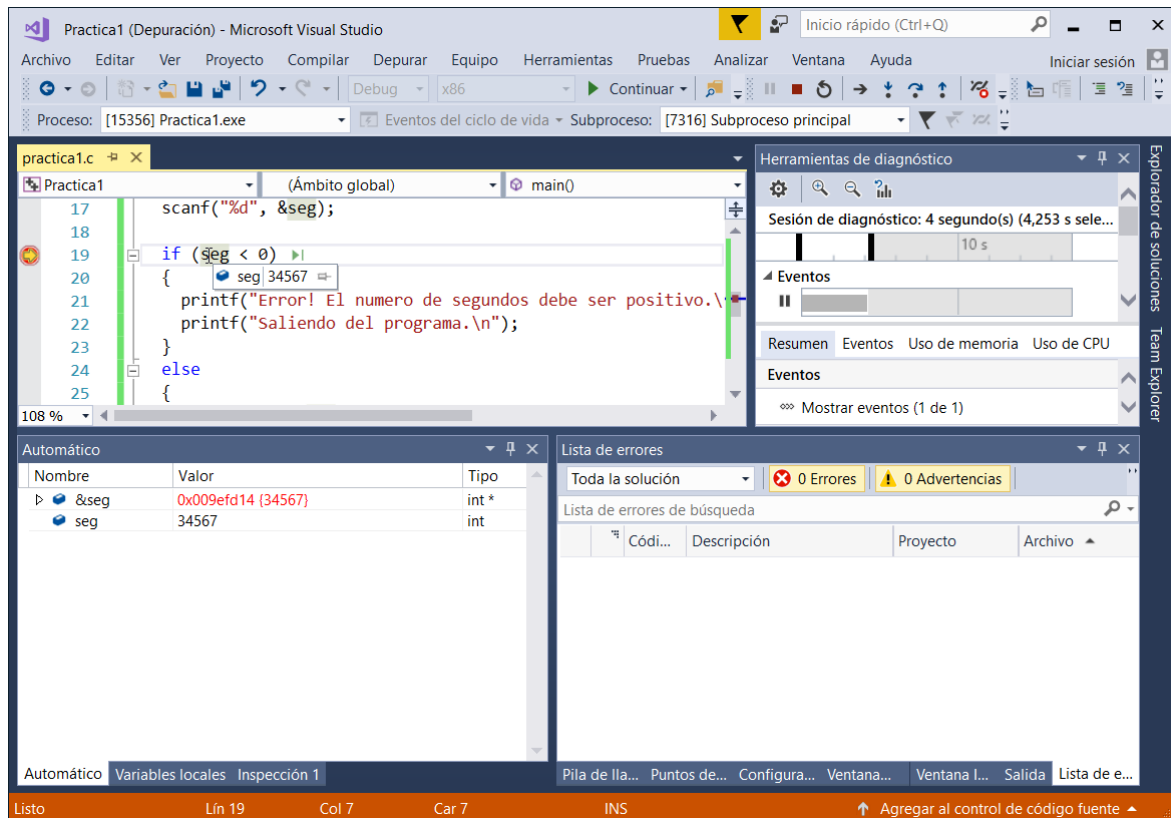
Para establecer un punto de parada (también llamado punto de ruptura ó *breakpoint*) en una línea, situamos el cursor en dicha línea y pulsamos F9 (o bien haciendo clic con el botón izquierdo del ratón en la parte izquierda de la sentencia donde queremos poner un punto de parada).

En nuestro ejemplo vamos a establecer un *punto de parada* en la línea de código 19:



Observe que se señala con un punto rojo (*parar/stop*). Eso significa que hemos establecido un *punto de parada* en dicha línea y que el programa detendrá su ejecución cuando llegue a dicha línea, antes de ejecutarla. Si se vuelve a pulsar F9 (o clic con el ratón), el *punto de parada* se elimina.

Iniciemos ahora al programa (F5) e introduzcamos el dato solicitado, por ejemplo, 34567 y pulsamos *Entrar*. Observamos que la ejecución se ha detenido en el punto de parada establecido. Se puede observar también que aparecen nuevos paneles de información en la parte inferior del entorno de desarrollo: *Automático*, *Variables locales*, etc. (mire las pestañas que aparecen en el fondo de la ventana) en los que puede ver, por ejemplo, los valores que van tomando las variables, se pueden escribir expresiones para que el depurador calcule su valor, etc. Su utilidad se verá según se progresa con el ejemplo.



Volvemos ahora al programa, que está todavía ejecutándose, pero detenido en el punto de parada establecido, según se puede ver en la figura anterior. El programa se detiene antes de ejecutar la línea de código.

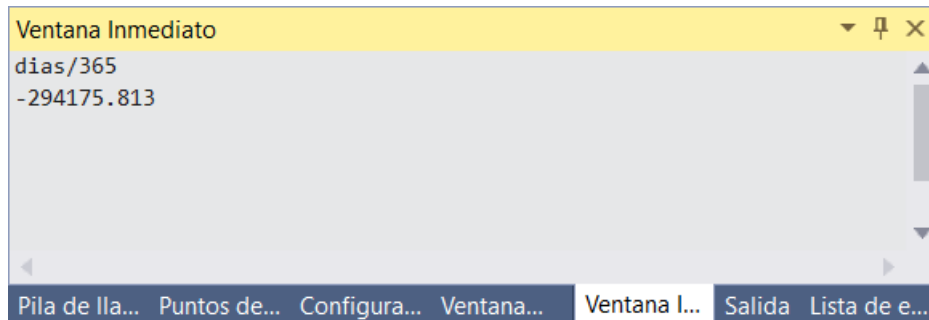
Si ahora cambia al panel *Variables locales*, haciendo clic en la pestaña *Variables locales*, observará que se muestran los valores que en este momento tienen las variables *días*, *horas*, *min* y *seg*:

Variables locales		
Nombre	Valor	Tipo
días	-107374176.	float
horas	-107374176.	float
min	-107374176.	float
seg	34567	int

Son las variables locales a la función en la que nos encontramos (en este caso, *main*). También se puede observar que, si se sitúa el puntero del ratón sobre el nombre de una variable, automáticamente se muestra el valor de dicha variable.

Como puede verse, el valor de *seg* es 34567, que es el que tecleamos. La variable *min* y variable *dias* tienen un valor sin sentido (basura) porque aún no han sido iniciadas con ningún valor.

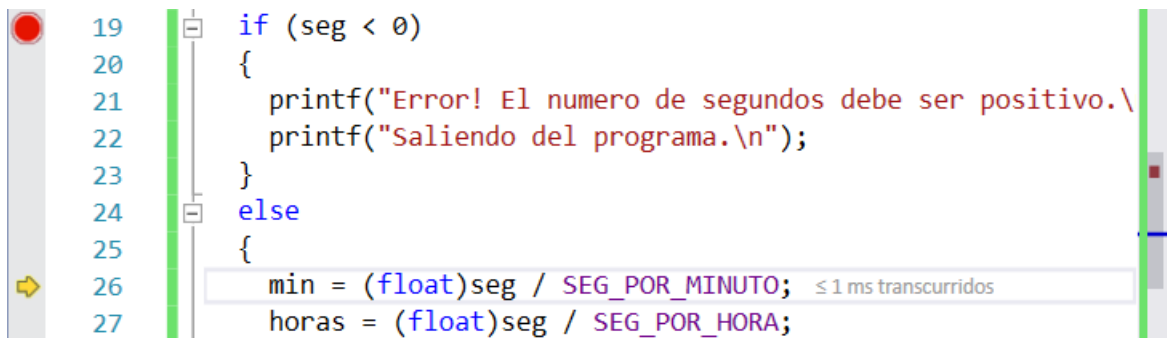
A continuación, podemos añadir una expresión para que Visual C++ calcule su valor en el momento, basada en el estado del programa y sus variables. Por ejemplo, podemos querer calcular cuánto vale *dias/365*, para obtener a cuantos años equivalen los segundos que hemos introducido. Para ello seleccionamos la pestaña *Inmediato* y tecleamos *dias/365* seguido de *Entrar* (si este panel no está, puede añadirlo ejecutando *Depurar > Ventanas > Inmediato*):



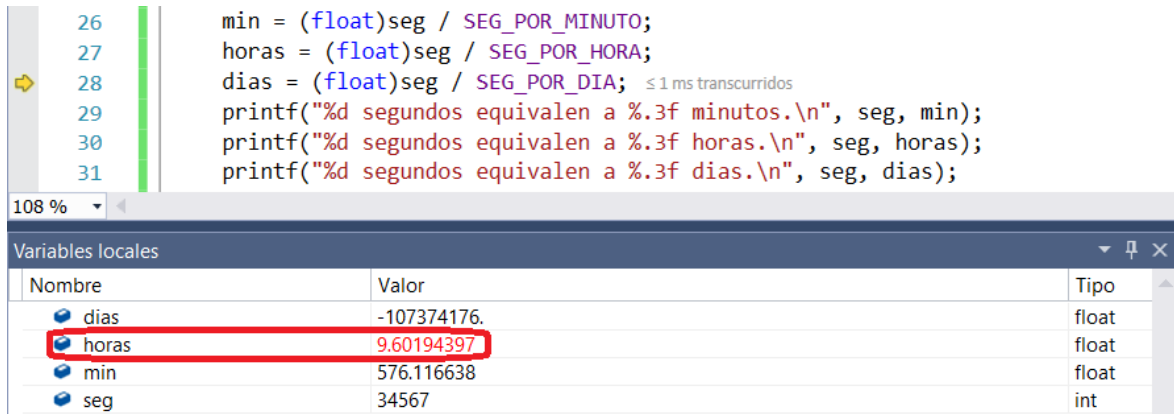
El valor obtenido no tiene sentido ahora, ya que la variable *dias* todavía no tiene un valor válido (tiene basura).

También pueden establecerse varios puntos de parada y saltar de uno a otro pulsando sucesivamente F5.

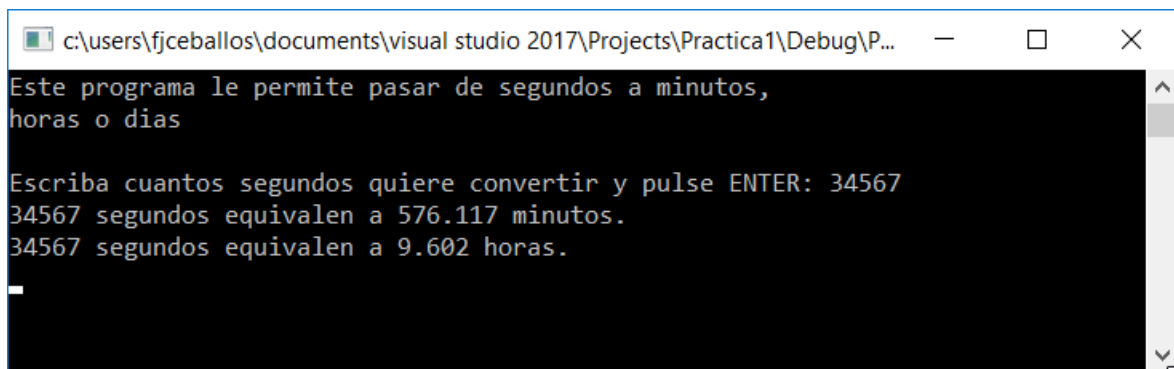
A continuación, vamos a emplear otra facilidad muy interesante del depurador, que es la *ejecución paso a paso*. Igual que el depurador es capaz de detenerse en una línea concreta, también es capaz de ejecutar el código, línea a línea, deteniéndose cada vez en la siguiente instrucción a ejecutar. Para ello basta con pulsar la tecla F10 repetidamente. Por ejemplo, la pulsamos una vez y observamos cómo la flecha amarilla avanza desde la línea 19 hasta la línea 26:



¿Qué ha ocurrido? Se ha ejecutado la sentencia *if (condición)* y como la condición *seg < 0* no se ha cumplido (*seg* no es menor que cero), el flujo del programa ha saltado hasta la primera línea de código del bloque que sigue a la cláusula *else* (si no se cumple), que es la siguiente sentencia a ejecutar (la 26, la que está con la flecha amarilla a la izquierda). Si pulsamos F10 dos veces más, observamos cómo la ejecución del programa avanza, y a su vez las variables *min* y *horas* actualizan su valor. Observar que el valor de *horas* aparece en color rojo; esto indica que este valor acaba de ser modificado en el último paso.

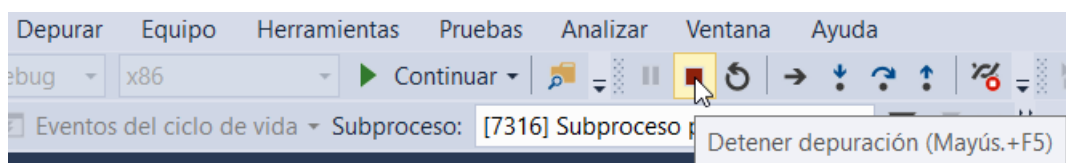


Si continuamos ejecutando paso a paso (por ejemplo, hasta la línea 31)) y observamos simultáneamente la consola dónde se está ejecutando la aplicación, observaremos cómo van apareciendo en la misma los resultados según se ejecutan las sentencias *printf*:



Podemos pulsar en cualquier momento F5 y el programa continuará hasta el final o hasta el siguiente punto de parada, si lo hubiere.

Puede abandonar la depuración en cualquier momento que desee pulsando *Mayúsculas + F5* o haciendo clic en el botón correspondiente de la barra de herramientas:



El depurador tiene muchas otras opciones en las que no entraremos en este momento, pero que usted mismo puede ir descubriendo. Por ejemplo, ¿qué diferencia hay entre utilizar la tecla *F10* o la tecla *F11* en un proceso de depuración?

Como norma general **cada vez que vayamos a trabajar en un programa nuevo debemos cerrar el proyecto actual**. Para ello ejecutaremos la opción *Cerrar solución* del menú *Archivo*.

7. Ejercicios a realizar

Una vez se haya habituado a las tareas descritas en los puntos anteriores, realice los siguientes ejercicios.

1. Depure el programa de nuevo, pero esta vez introduciendo un número de segundos negativo. Observe qué ocurre cuando se ejecuta la sentencia *if*.
2. Cree un nuevo proyecto y un nuevo archivo fuente llamado *divis.c* con el siguiente programa:

```
#define _CRT_SECURE_NO_DEPRECATED
#include <stdio.h>
#include <stdlib.h>

int main ()
{
    int max, num, resto;

    printf ("Este programa imprime los numeros divisibles\n");
    printf ("por 7 menores que numero dado.\n\n");

    printf ("Introduzca el numero: ");
    scanf ("%d", &max);

    if (max <= 0)
    {
        printf ("El numero debe ser positivo!!\n");
    }
    else
    {
        for (num=1; num <= max; num++)
        {
            resto = num % 7;

            if (resto == 0)
                printf ("%d es divisible por 7\n", num);
        }
    }

    system ("pause");
    return 0;
}
```

1. Compile, enlace y ejecute el programa.
2. Establezca un *punto de parada* en la línea *resto = num %7;* y pulse F5 para depurar el programa. Observe las variables y pulse de nuevo F5, para continuar la ejecución. ¿Qué ocurre?
3. Depure el programa desde el principio, pulsando F10, y siga línea a línea su ejecución. Observe atentamente los valores que toma la variable *num* a partir de la sentencia *for*. Trate de deducir qué hace y cómo se comporta la sentencia *for*.
4. Observe los valores que va tomando la variable *resto*. Trate de deducir qué operación realiza el operador *%*.