Microsoft Visual C++ 6.0

MANUAL SIMPLIFICADO DEL USUARIO

1. Entorno de desarrollo de Visual C++

1.1 Iniciar Visual C++

Para iniciar el entorno de desarrollo de *Visual C++*:

a. En el escritorio de Windows, sitúe el puntero del ratón en el menú *Inicio*, en la esquina inferior izquierda, y pulse el botón izquierdo del ratón.

b. Elija la opción *Programas* en el menú *Inicio*.

c. En el siguiente menú, encontrará la opción *Microsoft Visual C++ 6.0*, sitúese sobre ella con el ratón.

d. Elija la opción *Microsoft Visual C++ 6.0* en el menú.

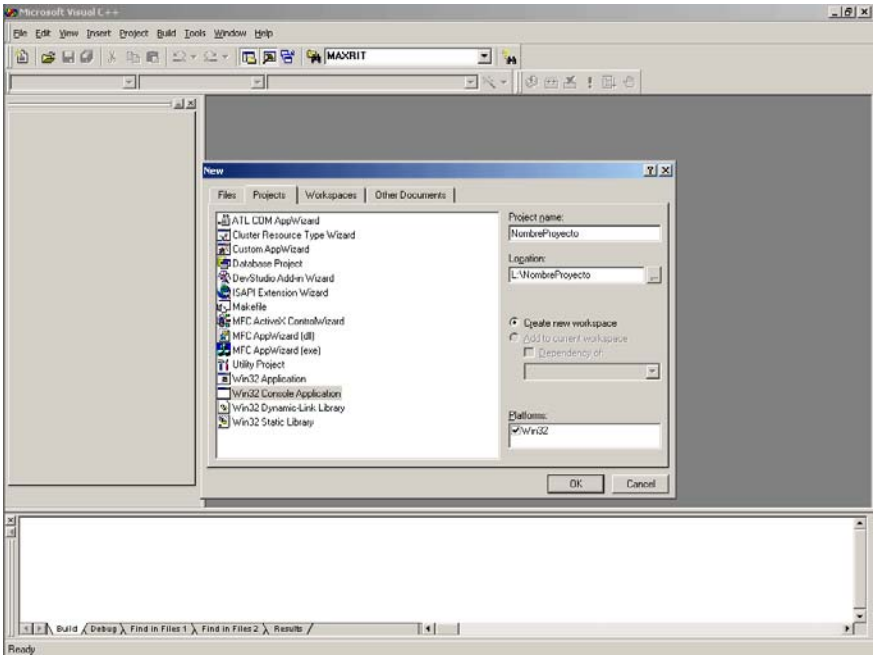
e. Transcurrido el tiempo de carga, se muestra el escritorio de *Microsoft Visual C++ 6.0*. Cierre la ventana adicional que se presenta, *Tip of the day*, pulsando el botón *Close*.

1.2 Crear una nueva aplicación

Al principio, el área de trabajo del entorno de desarrollo está vacía. Para crear ahora la estructura de una nueva aplicación, proceda del modo siguiente:

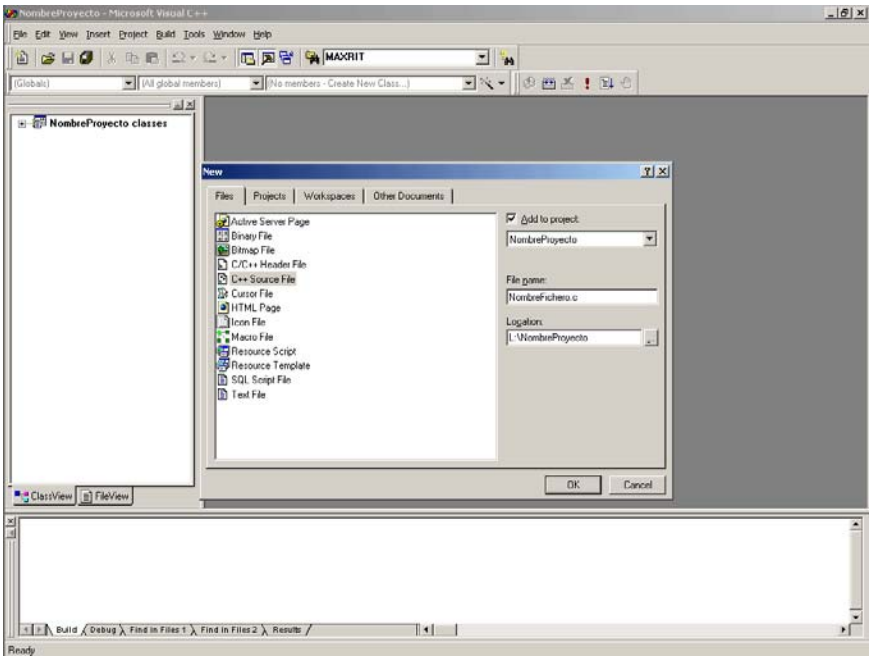
a. Elija la opción *New* en el menú *File*.

b. A continuación, se presenta una ventana en la que puede elegir qué tipo de aplicación desea crear. Seleccione la ficha *Projects*. En nuestro caso la aplicación será del tipo *Win32 Console Application*. Por lo tanto, seleccione esta opción y escriba el nombre que desee para su aplicación en el cuadro *Project name*. Escriba el nombre que desee para su aplicación. En el cuadro *Location*, seleccionar la ruta de acceso a su directorio (generalmente la **L:**). Automáticamente, se crea una ruta de acceso para el lugar donde se guardarán los archivos del programa. Confirme el cuadro de diálogo pulsando *OK*. Seleccione *An empty Project* y *Finish*. Finalmente se pulsa *OK* y se ha creado un proyecto nuevo.



c. El paso siguiente es crear un fichero y añadirlo al proyecto actualmente vacío. En la opción

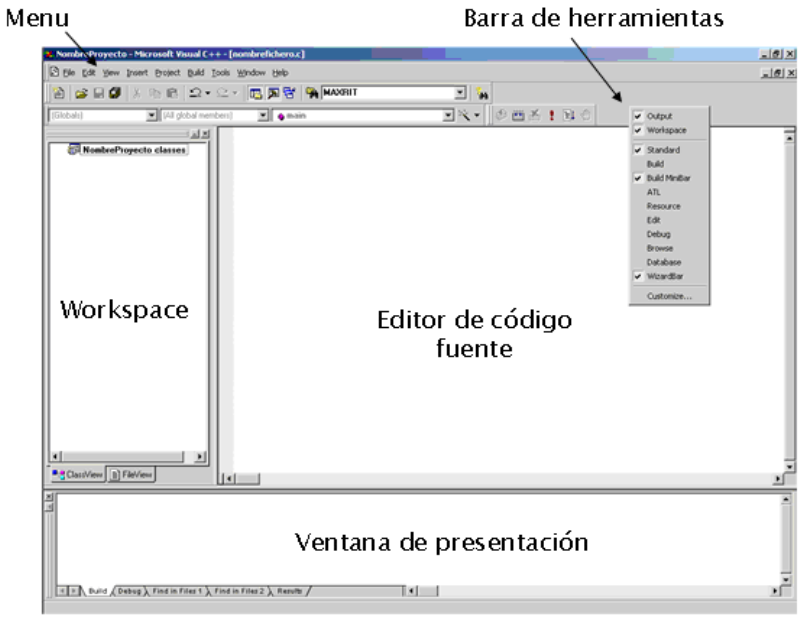
New del menú *File*, seleccione *C++ Source File* para crear un fichero de texto que contenga el código fuente. Asigne un nombre a este fichero fuente en el campo *File name*. La extensión de este fichero debe ser necesariamente `.c`. Confirme pulsando *OK*.



d. El entorno ya esta listo para introducir el código.

A continuación se describirán algunos de los componentes del entorno de desarrollo y sus propiedades más importantes.

2. Los complementos del entorno de desarrollo



2.1 Menu

Microsoft Visual C++ 6.0 tiene un menú dinámico que puede adaptarse a los diferentes estados del entorno de desarrollo.

- El menú *File* dispone de algunas opciones para los últimos archivos o proyectos abiertos.
- El menú *Tools* se puede adaptar a las necesidades de cada usuario mediante la opción *Customize*.
- El menú *Help* está dedicado al sistema de ayuda de Visual C++.

Muchas opciones de menú son activables a través de combinaciones de teclas. La configuración de dichas combinaciones se puede personalizar en el menú *Tools*, opción *Customize*, seleccionando la ficha *Keyboard* del cuadro de diálogo. Esta personalización no es recomendable en los laboratorios de la universidad.

2.2 Barras de herramientas

Las barras de herramientas se pueden considerar como un atajo de las opciones de menú. Pulsando con el botón derecho del ratón en un espacio libre de las barras de herramientas existentes, puede mostrar nuevas barras seleccionando la opción deseada. Así, puede mostrar otra barra que nos facilite el acceso inmediato a opciones de compilación (*Build*), depuración de errores (*Debug*), etc.

La opción *Customize* permite modificar la estructura de las mismas, principalmente:

- Agregar un botón a la barra de herramientas: Dentro de la ficha *Commands*, seleccione la nueva función en el cuadro de lista *Categories*. Pulse a continuación el botón que desee (cuadro *Buttons*) y lea la descripción del campo *Description*. Cuando ambos coincidan, arrastre el botón a la barra de herramientas que desee.
- Eliminar un botón de la barra de herramientas: Arrastre el botón que quiere eliminar fuera de la

barra de herramientas y llévelo al cuadro de diálogo *Customize*. El botón eliminado queda incluido automáticamente en la categoría correspondiente.

Realizadas las modificaciones, puede cerrar el cuadro de diálogo pulsando *Close*.

2.3 Workspace

Para el trabajo con Microsoft Visual C++, *Workspace* es una herramienta básica. Puede contener varias fichas. Para seleccionarlas, se utilizan los botones de la parte inferior de Workspace.

Cuando no hay abierto ningún proyecto, Workspace consta de una sola ficha: *InfoView*. Infoview sirve para representar la estructura del manual en pantalla. Las otras fichas sólo se muestran cuando hay un proyecto abierto. Al crear un nuevo proyecto, se activa una ficha nueva en Workspace: *ClassView*. ClassView permite manejar de forma sencilla los elementos más importantes del entorno de desarrollo: las subrutinas y funciones.

Los detalles acerca de las mismas se pueden obtener en pantalla pulsando el signo + que hay junto a cada una de ellas, o bien, pulsando dos veces con el ratón sobre el elemento correspondiente de la lista. Las funciones y variables de definición global se muestran al ampliar la opción *Globals* pulsando el signo +.

Una propiedad interesante de ClassView es la de permitir modificaciones en el código fuente con tan sólo pulsar dos veces en la función deseada. Las opciones representadas en ClassView no están guardadas en un fichero específico, sino que se van leyendo desde los diferentes archivos de código fuente. Este sistema permite realizar modificaciones manuales en el código fuente que ClassView incorpora después automáticamente.

Por último, mencionar la ficha *FileView* de Workspace. FileView muestra diferentes archivos que, o bien pertenecen al proyecto abierto, o de algún modo son decisivos para el mismo. Representa la conexión lógica de todos los archivos. Los primeros que se muestran son los que pertenecen al proyecto. En la subcarpeta *Dependencies* se encuentran archivos de los que el proyecto depende de algún modo (por ejemplo archivos de extensión `.h`).

2.4 Editor de código fuente

Posee algunas ampliaciones que simplifican considerablemente el manejo de los códigos fuente y, por tanto, la programación:

- Espacios en blanco virtuales: Al pulsar la tecla **Entrar** el cursor se sitúa de nuevo en la posición en la que se ha comenzado a escribir. No debe utilizar tabuladores, esta opción le permite diseñar un espaciado concreto en su programa.
- A los bloques entre llaves, que forman una unidad, se les aplica automáticamente una sangría.
- El cuadro de diálogo *Go To* facilita la navegación por el código fuente.
- A los pasajes del código fuente utilizados con mayor frecuencia, se les puede asignar marcadores de texto para permitir el acceso directo.
- Se resalta la sintaxis de C++ marcando en color las palabras clave.

2.5 Ventana de presentación

Aparece seleccionando la opción *Output* de entre las que aparecen al pulsar con el botón derecho del ratón en un sitio libre de las barras de herramientas. Consta también de varias fichas que se pueden seleccionar a través de los botones de la parte inferior:

- La ficha *Build* proporciona mensajes de estado del compilador de Visual C++, así como sus mensajes de error. Indica el nombre del archivo, el número de página y el número del error, así como una descripción del mismo. Para alcanzar la posición en que está localizado el error, puede pulsar dos veces el mensaje de error o bien utilizar el menú contextual de la ficha Build. Para esto último, debe seleccionar la opción *Go To Error/Tag*.
- La ficha *Debug* muestra los mensajes emitidos por el depurador.
- Seleccionando la ficha *Find in Files*, en la ventana de presentación se muestran los resultados de la última búsqueda de texto en varios archivos.

La opción *Hide* del menú contextual permite cerrar la ventana de presentación. Para verla de nuevo, también puede seleccionar *View/Output*.

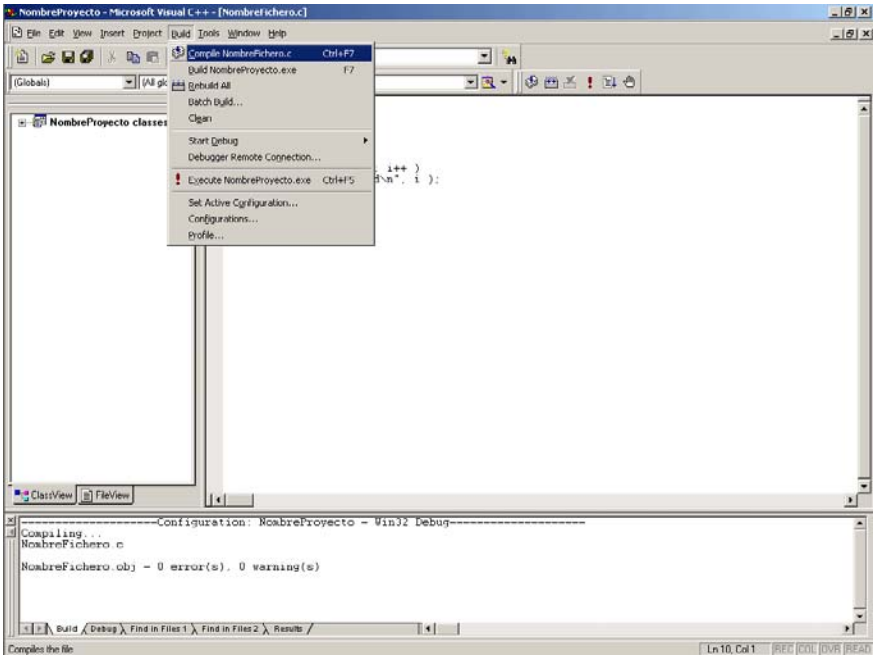
3. Compilador y vinculador

Antes de iniciar el programa, es necesario explicar algunos aspectos del proceso de compilación. La compilación de un programa no es un proceso homogéneo, sino más bien una sucesión de diferentes operaciones, cuya comprensión puede ser muy útil.

3.1 La compilación

En primer lugar, el **compilador** compila el archivo de código fuente. Este proceso de compilación afecta también a los archivos de encabezamiento (con la extensión `.h`); los archivos de código fuente incorporan estos archivos de encabezamiento a través de las instrucciones `#include`, de modo que éstos también se compilan. El resultado de este proceso de compilación es un así llamado archivo de objeto, con la extensión `.obj`. Para cada archivo de código fuente, se crea un archivo de objeto de igual nombre.

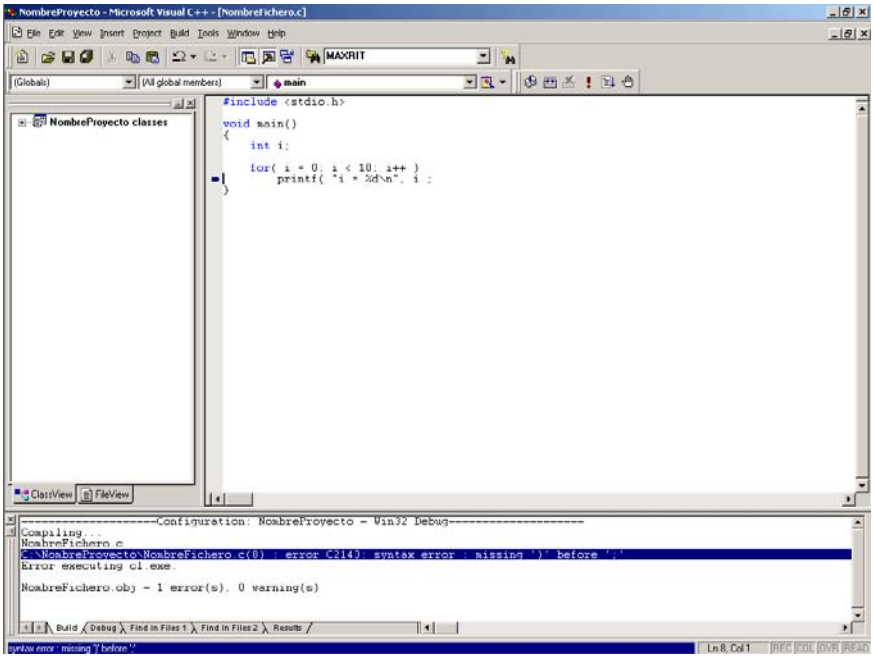
En el menú *Build*, con la opción *Compile NombreFichero.c* se activa la ejecución del proceso de compilación.



También se puede compilar con la combinación de teclas **CTRL** y **F7**.

Puede seguir cada paso en la ventana de presentación. Aquí se indicaran los errores (**error(s)**) y los avisos de posibles errores (**warning(s)**). Solo se podrá pasar al siguiente paso si el número de errores es cero. Los warnings son avisos que algo podría estar escrito mal. Es recomendable verificarlos.

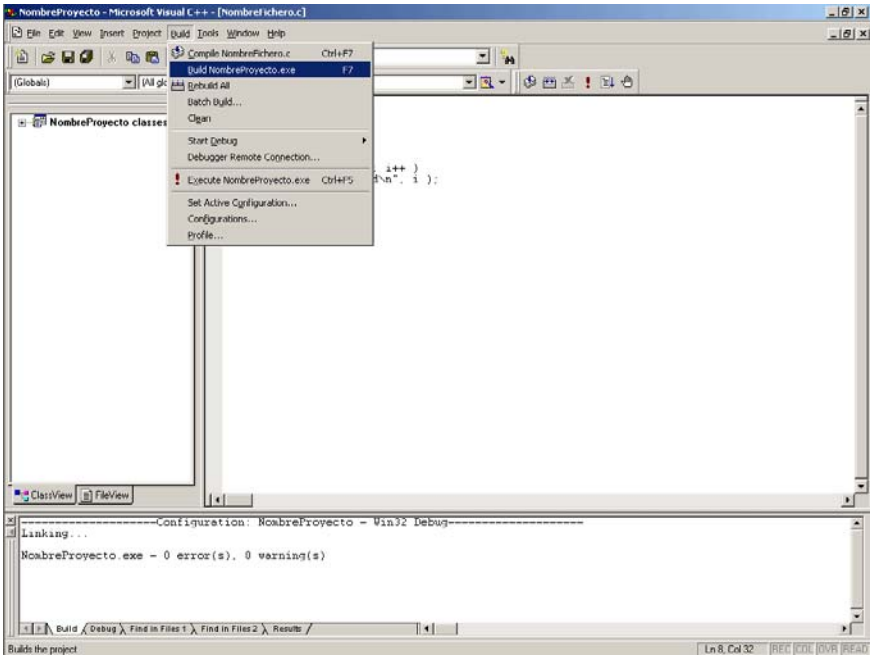
En la figura siguiente se ve un ejemplo de error. La línea que aparece en la ventana de presentación da una información sobre la línea de código donde se encuentra este error `(8)` (línea 8) y que el error es por la falta del paréntesis `missing ')'`. Pulsando dos veces en el mensaje de error, puede acceder directamente a la parte del programa que ha provocado dicho error.



3.2 El proceso de vinculación

Una vez compilados todos los archivos de código fuente (`.c` y `.h`), ya se pueden agrupar todos ellos en un único archivo. De esta tarea se ocupa el l**inker**. Recoge cada uno de los elementos básicos y crea un archivo `.exe` ejecutable.

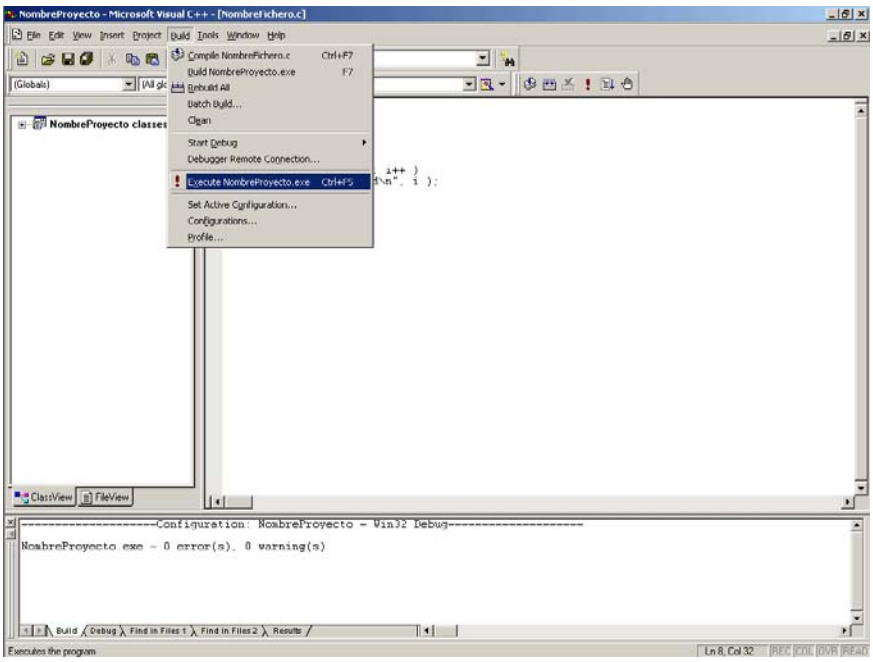
En el menú *Build*, con la opción *Build NombreProyecto.exe* se activa la ejecución del proceso de vinculación.



También se puede compilar con la tecla **F7**.

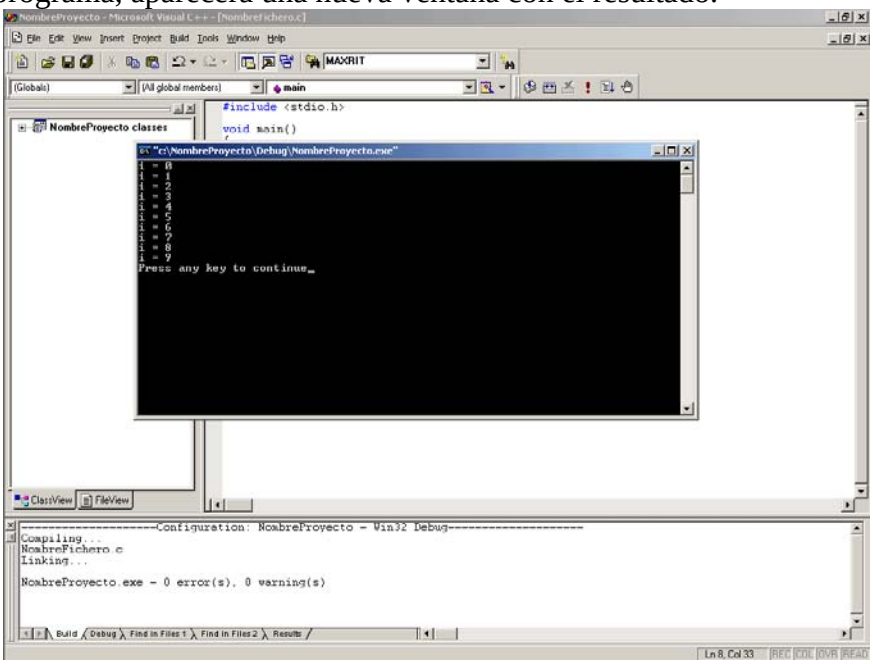
3.3 Ejecutar el programa

Ahora ya puede ejecutar el programa. En el menú *Build*, con la opción *Execute NombreProyecto.exe* se activa la ejecución del proceso de vinculación.



También se puede compilar con la combinación de teclas **CTRL** y **F5**.

Ejecutando el programa, aparecerá una nueva ventana con el resultado.



Al finalizar la ejecución habrá que pulsar una tecla cualquiera para cerrar la ventana del resultado.

3.4 Atajos

Las operaciones de compilación y vinculación se pueden hacer en un paso solo con la opción *Rebuild All* del menú *Build* que incluirá los **Warnings** en cada proceso de compilación efectuado. Existen, sin embargo, otras alternativas:

Se puede elegir la opción *Execute Proyecto.exe* del menú *Build*, aunque el programa no esté compilado ni vinculado. En este caso, se le solicitará si quiere iniciar el proceso de compilación (la expresión build representa, en este caso, compilación y enlace a la vez), puesto que el archivo

NombreProyecto.exe no existe todavía. Si responde **Yes**, comienza el proceso de compilación.

Puede seguir cada paso en la ventana inferior del área de trabajo. En primer lugar, se compilan los diferentes archivos y, finalmente, son agrupados por el vinculador (linker). Después, la ventana del programa se presenta en pantalla. Si obtiene algún mensaje de error durante el proceso de compilación deberá comprobar si ha cometido algún error al introducir alguna parte del texto.

3.5 Los diferentes modos de compilación

Visual C++ diferencia entre dos modos de compilación: el modo *Debug* y el modo *Release*.

En nuestro caso, el proyecto creado se debe compilar en el modo Debug como aplicación de Windows. Si selecciona Debug, durante la compilación se guardan determinadas informaciones en los archivos de objeto y en los archivos ejecutables. De este modo se pueden detectar a continuación los posibles errores del programa con el depurador de errores. Sin estas informaciones guardadas, el depurador no puede realizar su trabajo.

Su archivo de código fuente lo encontrará directamente en la carpeta de proyecto. Si realiza una compilación del programa, automáticamente se crea un directorio de la carpeta de proyecto, para los archivos compilados. Según el modo de compilación, este directorio se llamará Debug o Release. Su preocupación en adelante se centrará en el modo de compilación *Debug*.

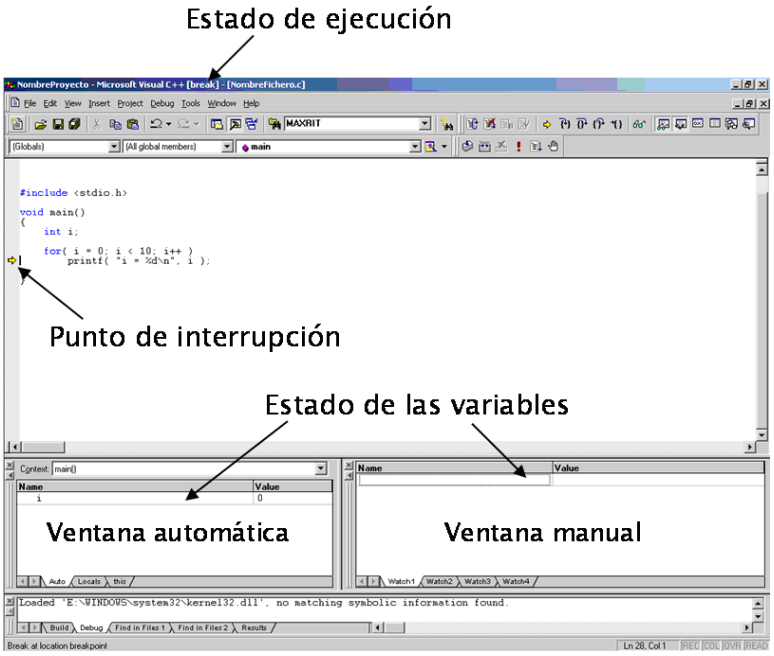
4. Depuración de errores

El depurador de errores es una herramienta que **ayuda el programador a encontrar los errores** algorítmicos (no los sintácticos). Eso significa que no corrige los errores por si solo sino permite una ejecución paso a paso del programa para ayudar la identificación del punto en que se verifica el error.

Todo depurador de errores ofrece una serie de funciones que se pueden dividir en:

- **Puntos de interrupción:** comunican al depurador en qué fragmento debe interrumpir la ejecución de un programa.
- **Presentación y modificación de variables:** si se ha detenido un programa, puede consultar y modificar los valores de las variables válidas.
- **Controles de la ejecución del programa:** puede ejecutar programas paso a paso y observar con exactitud cómo se desarrolla el programa.

En el modo de depuración de errores, el entorno presenta un aspecto diferente. A lado del nombre del fichero en la barra horizontal aparece el estado de ejecución del programa. En el ejemplo de la figura la palabra *break* indica que el programa está actualmente parado. Y en particular una flecha amarilla en el borde izquierdo del código fuente indica el punto de interrupción. Las dos ventas del final de la pantalla muestran los valores actuales de las variables usadas en el código.



Del modo de depuración de errores se puede salir (sin ejecutar el programa) en cualquier momento eligiendo la opción *Debug/Stop Debugging*, de la barra de herramientas. De igual manera, se puede elegir de ejecutar el programa hasta el final eligiendo la opción *Debug/Go*, de la barra de herramientas.

4.1 Puntos de interrupción (Breakpoints)

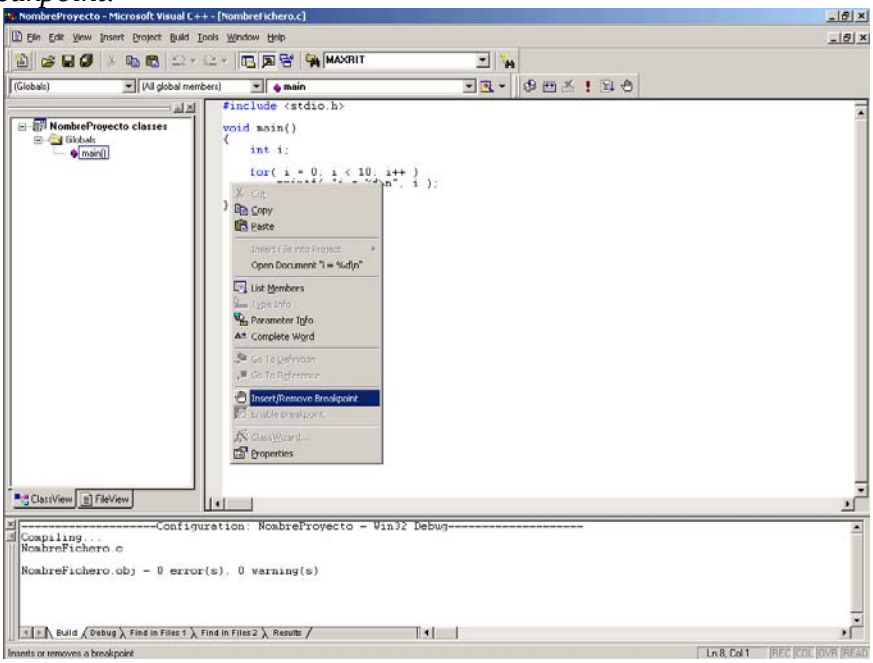
Los puntos de interrupción permiten ejecutar un programa hasta un determinado punto (elegido por el programador) y verificar si en dicho punto el programa se ha desarrollado correctamente (es decir los resultados son los esperados).

Se dispone de diferentes tipos de puntos de interrupción.

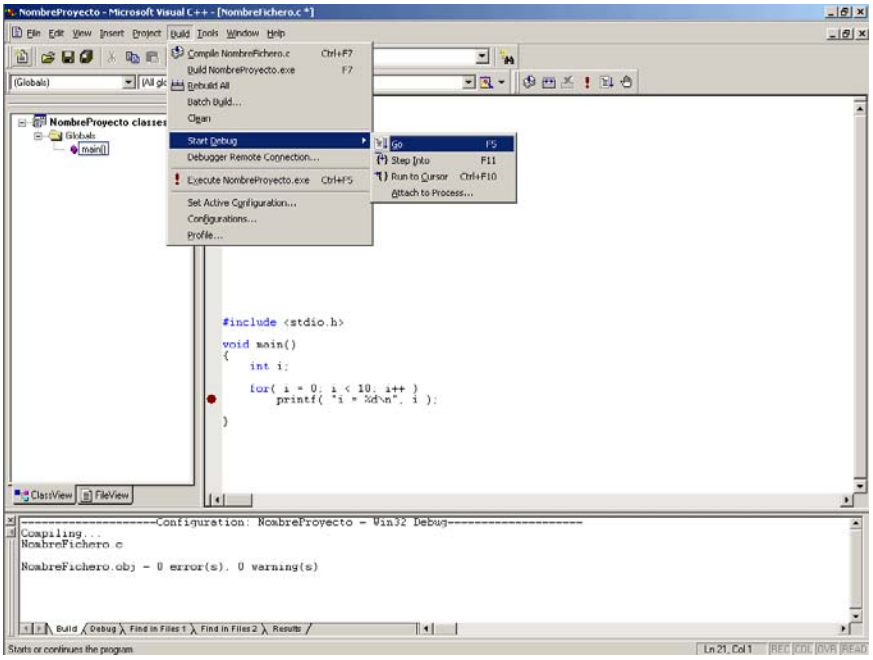
4.1.1 Puntos de interrupción vinculados a una línea de código

Puede definir un punto de interrupción vinculado a una línea de código, el cual siempre detendrá el desarrollo de un programa en una posición fija que se puede establecer en el código fuente.

Pulsando con el botón derecho del ratón en la línea de programa que desee encontrará la opción *Insert/Remove Breakpoint*.



Seleccionada dicha opción, un punto rojo representa el punto de interrupción en que se detendrá la ejecución del programa. Inicie la compilación y ejecución eligiendo la opción *Build/Start Debug/Go* o con la tecla **F5**.



A fin de reanudar la ejecución del programa sin el punto de interrupción, puede eliminar o bien desactivar dicho punto. Para eliminarlo, se utiliza la opción *Remove Breakpoint*. Para desactivarlo, *Disable Breakpoint*. Este último presenta la ventaja de que posteriormente se puede activar a través de la opción de menú *Enable Breakpoint*.

4.1.2 El comando *Run to Cursor.*

Este comando que se encuentra en el menú *Build/Start Debug*, es el tipo más sencillo y posee un carácter temporal. Se consigue que el programa se inicie, pero que luego se interrumpa su ejecución en la posición en la que se encontraba el cursor en el momento del inicio del programa. Tras elegir este comando basta con situar el cursor en el lugar en que se desee interrumpir la ejecución del programa.

Una vez efectuada la interrupción aparece una pequeña flecha amarilla junto a la línea donde antes se encontraba el cursor. Ahora puede optar entre finalizar la ejecución del programa (*Debug/Stop Debugging*, de la barra de herramientas) o reanudar el programa (*Build/Start Debug/Go*). Después de realizar una parada en el fragmento en cuestión, la aplicación se sigue desarrollando con normalidad.

4.1.3 Puntos de interrupción de datos

En este tipo de interrupciones, la ejecución de un programa se detiene cuando varía el valor de una variable especificada previamente.

Esto resulta útil, por ejemplo, cuando ha comprobado mediante puntos de interrupción "normales" que el valor de una variable ha cambiado, pero no puede localizar la causa de esa variación. Proceda del modo siguiente:

a. Abra el cuadro de diálogo de puntos de interrupción (en el menú *Edit*, la opción *Breakpoints*).
b. Elija la ficha *Data*.
c. En la línea de introducción, indique el nombre de la variable.
d. Si se trata de una variable local, se tiene que indicar en el cuadro de diálogo *Advanced*, que aparece al pulsar sobre la flecha que aparece en el cuadro *Break at*. Escriba el nombre de la función en la que se encuentra la variable en la línea de entrada *Function* del cuadro de diálogo *Advanced Breakpoint* y luego cierre dicho cuadro.
e. Cierre el cuadro de diálogo *Breakpoints* e inicie la aplicación con el comando *Build/Debug/Go*.

4.2 Ejecución de programas paso a paso

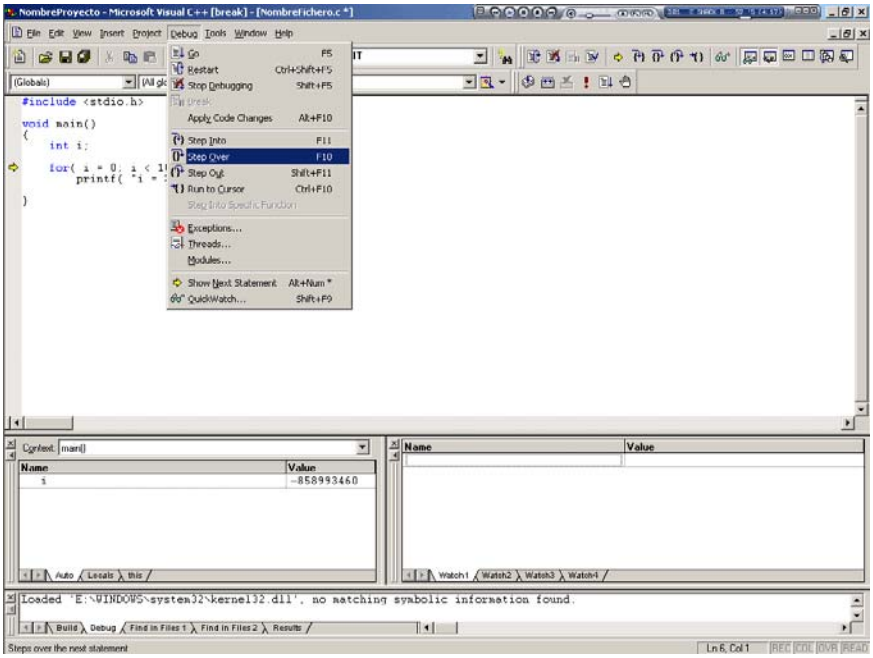
Es posible ejecutar un programa línea por línea mediante un depurador de errores.

4.2.1 Step over

La posibilidad más sencilla para ello es el comando *Step Over*, que se encuentra en el menú Debug cuando se interrumpe la ejecución del programa. Este comando provoca que se ejecute la línea de código fuente en la que se encuentra actualmente la flecha amarilla. Para ello debe definir un punto de interrupción vinculado a una línea, iniciar la ejecución del programa y elegir la opción Step Over del menú Debug o pulsar la tecla **F10**.

Tras estos pasos, la flecha amarilla que indica la posición del programa actual se encuentra junto a la siguiente línea de código fuente. Por lo tanto, desde el punto de interrupción se ha procesado exactamente una línea del código fuente.

Llegados a la ultima línea de código (generalmente la llave `}`), no tiene sentido ejecutar esta sentencia, por lo tanto se tiene la obligación de salir del modo de depuración de errores (*Debug/Stop Debugging*, de la barra de herramientas). En caso contrario (se ejecuta otro paso aunque estemos al final), aparecerá una nueva ventana que habrá que cerrar.



4.2.2 Step Into

El comando *Step Into*, salta hasta la función activada. Para ello debe definir un punto de interrupción vinculado a una línea, iniciar la ejecución del programa y elegir la opción Step Into del menú Debug o pulsar la tecla **F11**.

Esta opción sirve principalmente para **saltar dentro de la función apuntada por la flecha amarilla**. En el caso no se quiera ejecutar una función paso por paso sino solo se quiere saber su resultado, entonces se puede elegir la opción *Step Over* o pulsar la tecla **F10**.

4.2.3 Step Out

La última posibilidad para la ejecución paso a paso de programas es *Step Out*. Con este comando, puede continuar la ejecución del método actual hasta el final sin interrupción. El desarrollo del programa no se detiene de nuevo hasta que se regresa al método desde el que ha sido activado.

4.3 Presentación de valores de variables

En el modo de depuración existen varias ventanas que muestran los valores de las variables. Al interrumpir el programa, se pueden observar en la pantalla las ventanas siguientes:

4.3.1 La ventana (automática) de estado de las variables

Se encuentra en la parte inferior izquierda de la pantalla y está dividida en tres partes, que se pueden elegir a través de una ficha en el margen inferior. Tan sólo trabaje con:

- La ficha *Auto*. Muestra todas las variables que se han utilizado en la instrucción actual y en la anterior.
- La ficha *Locals*. Muestra todas las variables locales para la función actual.

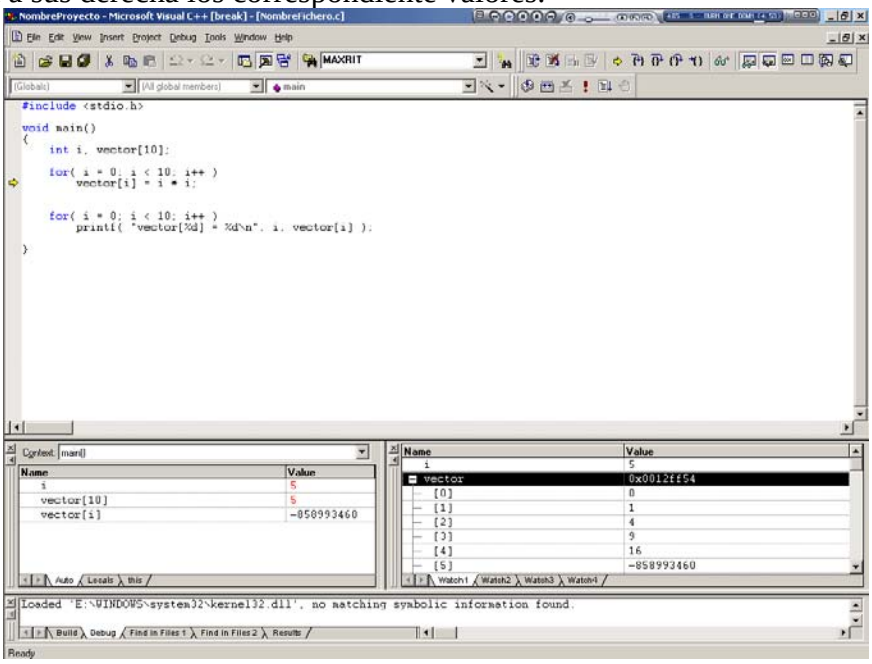
En ambas es posible obtener más información sobre algunas variables con ayuda de una pulsación en el signo correspondiente.

4.3.2 La ventana (manual) de estado de las variables o Watch

En la ventana de variables, no es posible controlar los valores de otras variables que no sean las mostradas automáticamente. Esto se puede realizar con la ventana *Watch*, que se encuentra a la derecha de la ventana de variables y posee una estructura similar.

Para comprobar otros valores de variables sitúe el enfoque en la ventana Watch mediante una pulsación con el ratón, elija la línea en la que se deben mostrar los nuevos valores de variables y escriba el identificador de variable en la línea actual. Tras confirmar con **ENTRAR**, el valor de las variables se muestra en la segunda columna de la tabla. Mediante "arrastrar" y "colocar" con el ratón también creará una línea con la variable correspondiente en la ventana Watch, de forma más sencilla.

En el ejemplo de la figura se ha querido conocer el valor de la variable `i` escribiéndola en la primera casilla disponible debajo de *Name*. Automáticamente a su derecha el depurador pone el valor actual de esta variable. De la misma manera, se ha insertado la variable `vector` (que es un vector) en la siguiente casilla. Siendo un vector, a su derecha aparece su dirección de memoria. Para visualizar los valores de los elementos del vector, hay que pulsar en el símbolo +. Debajo de la variable aparecerán los elementos con a sus derecha los correspondiente valores.

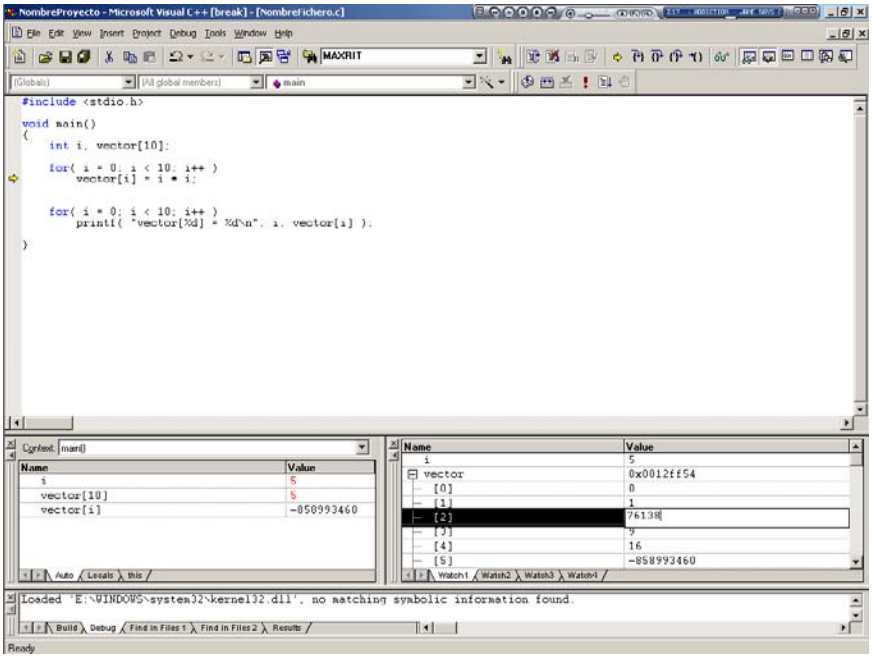


Es posible eliminar líneas de la ventana Watch a través del comando *Edit/Delete*. El orden de las líneas se puede modificar posteriormente mediante "arrastrar" y "colocar".

4.4 Modificación de valores de variables

Se permite modificar valores de variables en tiempo de ejecución del programa. Esto se puede realizar a través de la tabla que se encuentra en cada una de las ventanas realizando una doble pulsación con el ratón en un valor de variable. Con este método puede, por ejemplo, corregir primero de forma temporal el error detectado y comprobar así si con el valor modificado ha subsanado el error.

En el ejemplo se está modificando el valor del segundo elemento del vector asignando a esta variable el valor 76138.


NombreProyecto - Microsoft Visual C++ [break] - [NombreFichero.c]
File Edit View Insert Project Debug Tools Window Help
MAXRIT
(Globals)
(All global members)
main
#include <stdio.h>
void main()
{
int i, vector[10];
for(i = 0; i < 10; i++)
vector[i] = i * i;
for(i = 0; i < 10; i++)
printf("vector[%d] = %d\n", i, vector[i]);
}
Context: main()
Name
Value
i
5
vector[10]
5
vector[i]
-858993460
Auto
Locals
this
Name
Value
i
5
vector
0x0012ff54
[0]
0
[1]
1
[2]
76138
[3]
9
[4]
16
[5]
-858993460
Watch1
Watch2
Watch3
Watch4
Loaded 'E:\WINDOWS\system32\kernel32.dll', no matching symbolic information found.
Build
Debug
Find in Files 1
Find in Files 2
Results
Ready