

# Tutorial: Crear y usar una biblioteca personalizada de vínculos dinámicos (C++)

22/08/2019 • Tiempo de lectura: 12 minutos •   

## En este artículo

[Requisitos previos](#)

[Creación del proyecto DLL](#)

[Creación de una aplicación cliente que utiliza el archivo DLL](#)

[Vea también](#)

En este tutorial paso a paso se muestra cómo usar el IDE de Visual Studio para crear una biblioteca de vínculos dinámicos (DLL) propia escrita en Microsoft C++ (MSVC). Después, se muestra cómo usar la DLL desde otra aplicación de C++. Los archivos DLL (también conocidos como *bibliotecas compartidas* en los sistemas operativos basados en UNIX) son uno de los tipos de componentes de Windows más útiles. Se pueden usar como una manera de compartir recursos y código, y para reducir el tamaño de las aplicaciones. Los archivos DLL incluso pueden hacer que sea más sencillo dar servicio a las aplicaciones y ampliarlas.

En este tutorial, creará un archivo DLL que implementa algunas funciones matemáticas. Después, creará una aplicación de consola que usa las funciones del archivo DLL. También obtendrá una introducción a algunas de las técnicas de programación y a las convenciones que se usan en los archivos DLL de Windows.

En este tutorial se tratan las siguientes tareas:

- Cree un proyecto de DLL en Visual Studio.
- Agregue funciones exportadas y variables al archivo DLL.
- Cree un proyecto de aplicación de consola en Visual Studio.
- Utilice las funciones y variables que se importaron desde el archivo DLL en la aplicación de consola.
- Ejecute la aplicación completada.

Como sucede con una biblioteca vinculada estáticamente, un archivo DLL *exporta* variables, funciones y recursos por nombre. Una aplicación cliente *importa* los nombres para usar esas variables, funciones y recursos. A diferencia de una biblioteca vinculada estáticamente, Windows conecta las importaciones de la aplicación a las exportaciones de un archivo DLL en el tiempo de carga o de ejecución, en lugar de conectarse a ellos en el tiempo de vinculación. Windows requiere información adicional que no forma parte del modelo de compilación de C++ estándar para realizar estas conexiones. El compilador de MSVC implementa algunas extensiones específicas de Microsoft en C++ para proporcionar esta información adicional. Se explicarán estas extensiones según avanzamos.

Este tutorial crea dos soluciones de Visual Studio; uno que compila el archivo DLL y otro que compila la aplicación cliente. El archivo DLL usa la convención de llamada de C. Se puede llamar desde aplicaciones escritas en otros lenguajes de programación, siempre y cuando la plataforma, las convenciones de llamada y las convenciones de vinculación coincidan. La aplicación cliente usa la *vinculación implícita*, donde Windows vincula la aplicación al archivo DLL en el tiempo de carga. Esta vinculación permite que la aplicación llame a las funciones proporcionadas por el archivo DLL al igual que las funciones de una biblioteca vinculada de forma estática.

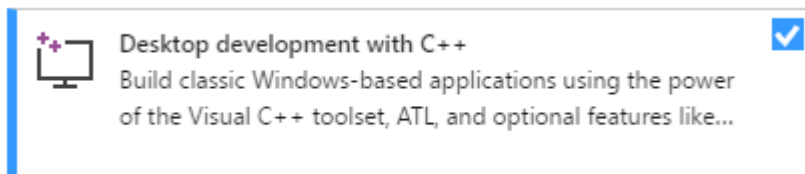
En este tutorial no se tratan algunas situaciones comunes. En el código no se muestra el uso de archivos DLL de C++ por parte de otros lenguajes de programación. No se muestra cómo [crear una DLL solo de recursos](#), ni cómo usar la [vinculación explícita](#) para cargar archivos DLL en tiempo de ejecución, en lugar de hacerlo en tiempo de carga. Puede usar MSVC y Visual Studio sin problemas para realizar todas estas operaciones.

Para consultar vínculos con más información sobre los archivos DLL, consulte [Crear archivos DLL de C o C++ en Visual Studio](#). Para obtener más información sobre la vinculación implícita y la explícita, vea [Determinación del método de vinculación que se va a usar](#). Para obtener información sobre cómo crear archivos DLL de C++ para usarse con lenguajes de programación que utilizan las convenciones de vinculación del lenguaje C, vea [Exportación de funciones de C++ para usarlas en ejecutables creados en C](#). Para obtener información sobre cómo crear archivos DLL para usarse con lenguajes de .NET, consulte [Llamar a funciones de un archivo DLL desde aplicaciones programadas en Visual Basic](#).

## Requisitos previos

- Un equipo que ejecuta Microsoft Windows 7 o versiones posteriores. Recomendamos Windows 10 para obtener la mejor experiencia de desarrollo.

- Una copia de Visual Studio. Para obtener información sobre cómo descargar e instalar Visual Studio, consulte [Instalación de Visual Studio](#). Al ejecutar el programa de instalación, asegúrese de que la carga de trabajo **Desarrollo para el escritorio con C++** está activada. No se preocupe si no ha instalado esta carga de trabajo al instalar Visual Studio. Puede ejecutar de nuevo el programa de instalación e instalarla ahora.



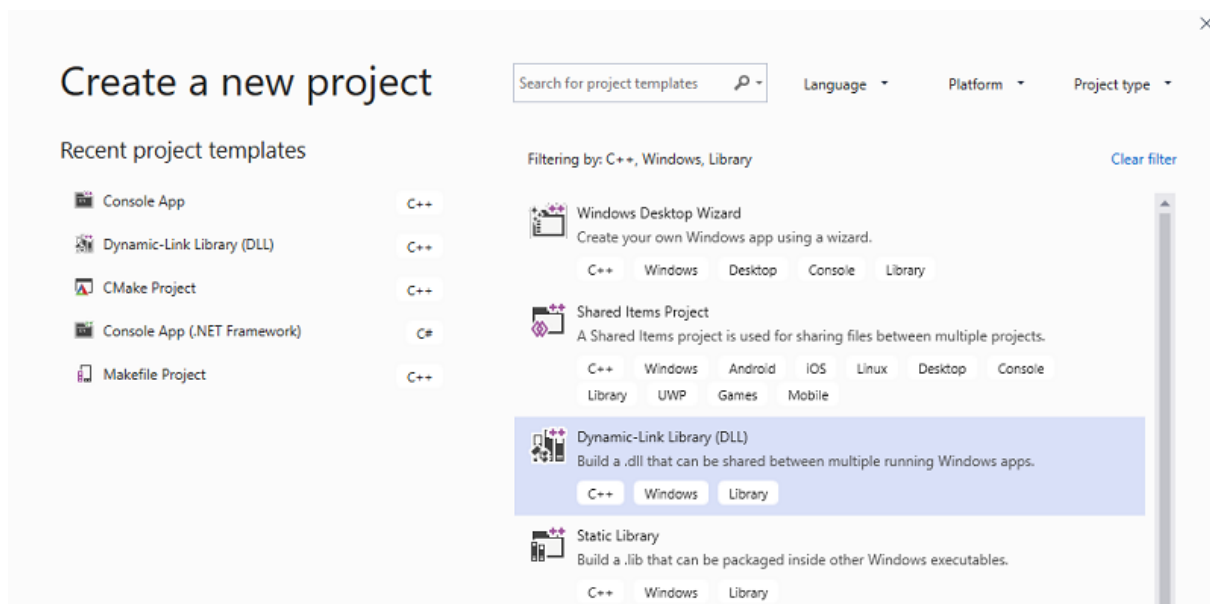
- Comprensión de los aspectos básicos del uso de IDE de Visual Studio. Si ha usado antes las aplicaciones de escritorio de Windows, probablemente esté al día. Para ver una introducción, consulte [Paseo por las características del IDE de Visual Studio](#).
- Conocer todos los fundamentos del lenguaje C++ para poder continuar. No se preocupe, no hacemos nada que sea muy complicado.

## Creación del proyecto DLL

En este conjunto de tareas, creará un proyecto para el archivo DLL, agregará el código y lo generará. Para comenzar, inicie el IDE de Visual Studio e inicie sesión si lo necesita. Las instrucciones varían ligeramente según la versión de Visual Studio que use. Asegúrese de que tiene la versión correcta seleccionada en el control de la parte superior izquierda de esta página.

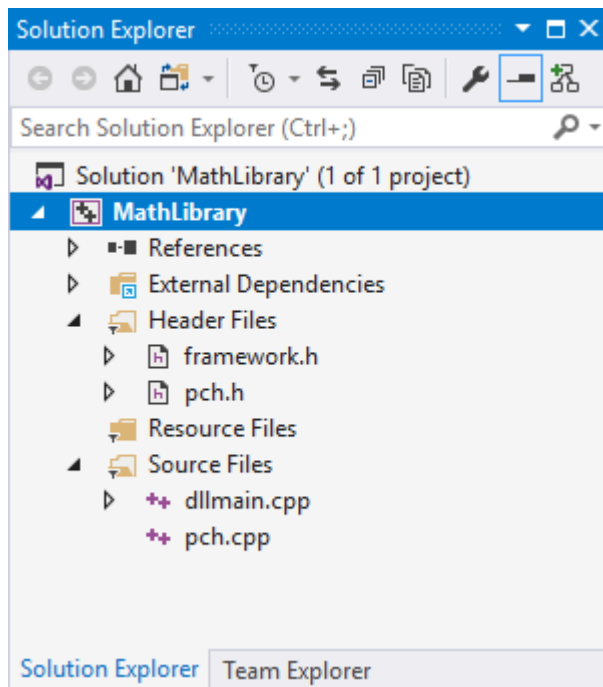
### Crear un proyecto de DLL en Visual Studio 2019

1. En la barra de menús, seleccione **Archivo > Nuevo > Proyecto** para abrir el cuadro de diálogo **Crear un nuevo proyecto**.



2. En la parte superior del cuadro de diálogo, establezca **Lenguaje** en **C++**, establezca **Plataforma** en **Windows** y establezca **Tipo de proyecto** en **Biblioteca**.
3. En la lista filtrada de tipos de proyecto, seleccione **Biblioteca de vínculos dinámicos (DLL)** y después **Siguiente**.
4. En la página **Configurar el nuevo proyecto**, escriba *MathLibrary* en el cuadro **Nombre del proyecto** para especificar un nombre para el proyecto. Mantenga los valores predeterminados de **Ubicación** y **Nombre de la solución**. Establezca **Solución** en **Crear nueva solución**. Desactive **Colocar la solución y el proyecto en el mismo directorio** si está activada.
5. Elija el botón **Crear** para crear el proyecto.

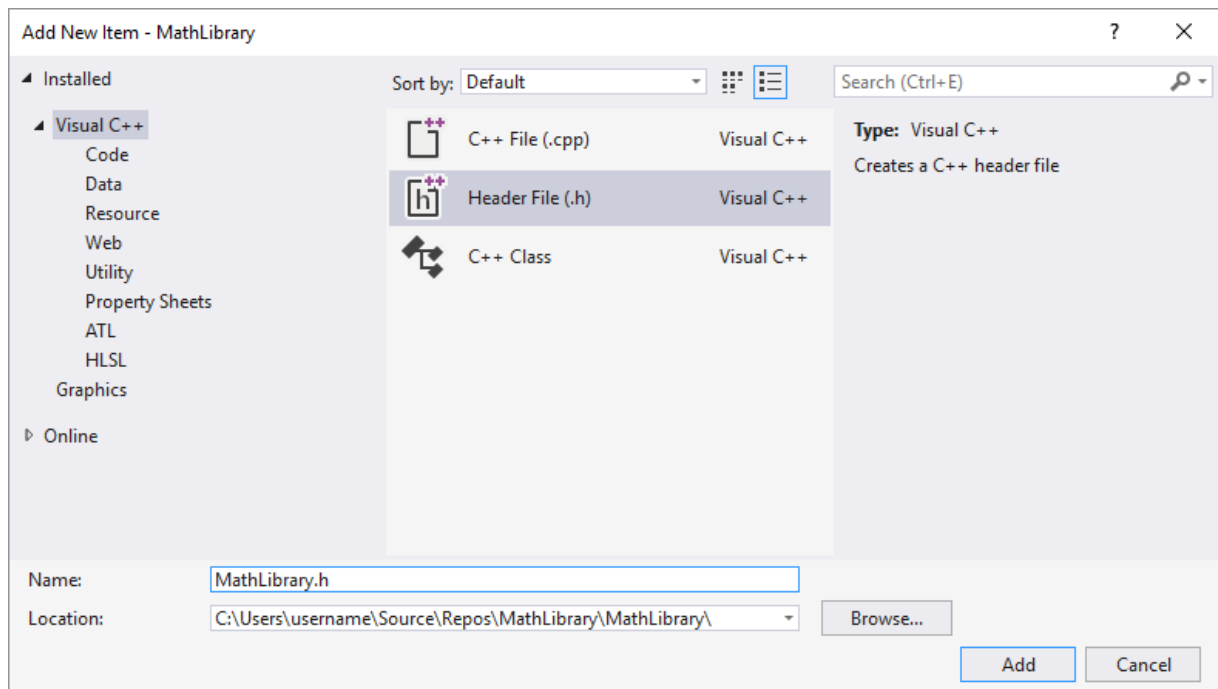
Una vez que se crea la solución, puede ver los archivos de origen y del proyecto generados en la ventana **Explorador de soluciones** de Visual Studio.



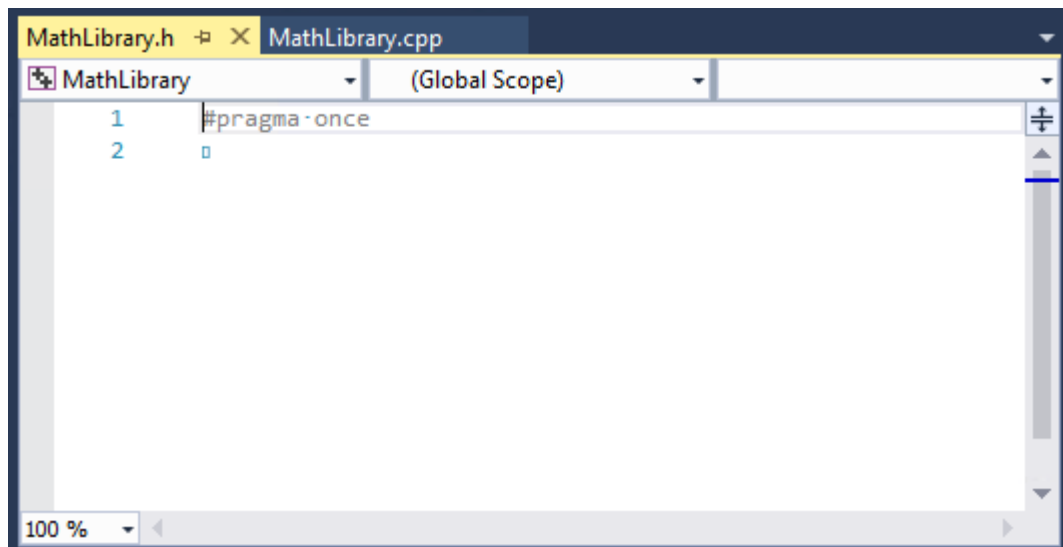
Ahora, este archivo DLL no sirve de mucho. Después, creará un archivo de encabezado para declarar las funciones de las exportaciones de DLL y, luego, agregará las definiciones de función al archivo DLL para que sea más útil.

## Agregar un archivo de encabezado al archivo DLL

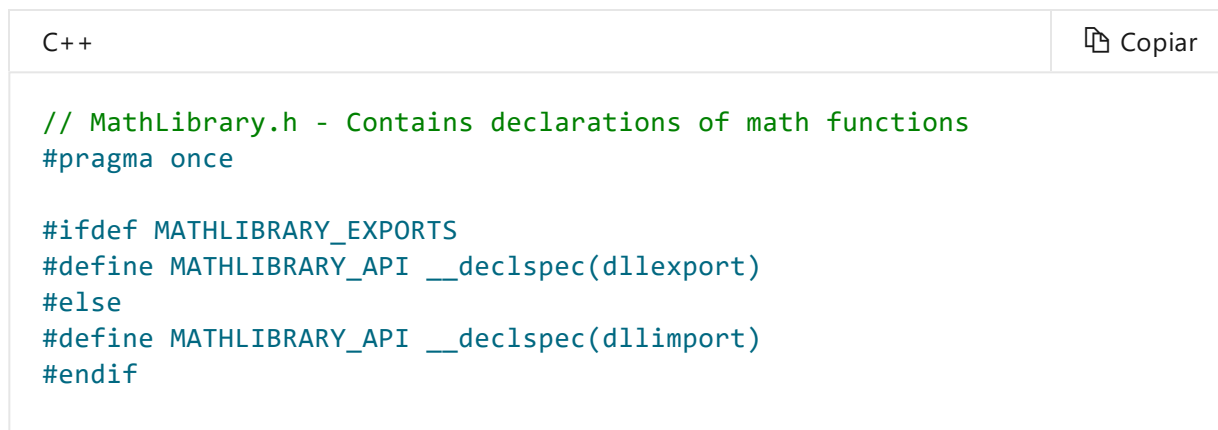
1. Para crear un archivo de encabezado para las funciones, en la barra de menús, elija **Proyecto > Agregar nuevo elemento**.
2. En el cuadro de diálogo **Agregar nuevo elemento**, en el panel izquierdo, seleccione **Visual C++** . En el panel central, seleccione **Archivo de encabezado (.h)** . Especifique *MathLibrary.h* como el nombre del archivo de encabezado.



3. Elija el botón **Agregar** para generar un archivo de encabezado en blanco, que se muestra en una nueva ventana del editor.



4. Reemplace el contenido del archivo por el código siguiente:



```
// The Fibonacci recurrence relation describes a sequence F
// where F(n) is { n = 0, a
//                { n = 1, b
//                { n > 1, F(n-2) + F(n-1)
// for some initial integral values a and b.
// If the sequence is initialized F(0) = 1, F(1) = 1,
// then this relation produces the well-known Fibonacci
// sequence: 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

// Initialize a Fibonacci relation sequence
// such that F(0) = a, F(1) = b.
// This function must be called before any other function.
extern "C" MATHLIBRARY_API void fibonacci_init(
    const unsigned long long a, const unsigned long long b);

// Produce the next value in the sequence.
// Returns true on success and updates current value and index;
// false on overflow, leaves current value and index unchanged.
extern "C" MATHLIBRARY_API bool fibonacci_next();

// Get the current value in the sequence.
extern "C" MATHLIBRARY_API unsigned long long fibonacci_current();

// Get the position of the current value in the sequence.
extern "C" MATHLIBRARY_API unsigned fibonacci_index();
```

Este archivo de encabezado declara algunas funciones para generar una secuencia de Fibonacci generalizada, dados dos valores iniciales. Una llamada a `fibonacci_init(1, 1)` genera la secuencia de números de Fibonacci familiar.

Observe las instrucciones de preprocesador en la parte superior del archivo. La plantilla de nuevo proyecto de un proyecto de DLL agrega **NOMBRE\_DE\_PROYECTO\_EXPORTS** a las macros de preprocesador definidas. En este ejemplo, Visual Studio define **MATHLIBRARY\_EXPORTS** cuando se compila el proyecto de DLL de MathLibrary.

Cuando la macro **MATHLIBRARY\_EXPORTS** está definida, la macro **MATHLIBRARY\_API** establece el modificador `__declspec(dllexport)` en las declaraciones de función. Este modificador indica al compilador y al enlazador que exporte una función o variable desde el archivo DLL para que se pueda usar en otras aplicaciones. Cuando **MATHLIBRARY\_EXPORTS** está definido, por ejemplo, cuando se incluye el archivo de encabezado de una aplicación cliente, **MATHLIBRARY\_API** aplica el modificador `__declspec(dllimport)` en las declaraciones. Este modificador optimiza la importación de la función o la variable en una aplicación. Para obtener más información, consulte [dllexport](#), [dllimport](#).

## Agregar la implementación al archivo DLL

1. En el **Explorador de soluciones**, haga clic con el botón derecho en el nodo **Archivos de código fuente** y elija **Agregar > Nuevo proyecto**. Cree un archivo .cpp denominado *MathLibrary.cpp*, de la misma manera que ha agregado un nuevo archivo de encabezado en el paso anterior.
2. En la ventana del editor, seleccione la pestaña de **MathLibrary.cpp** si ya se ha abierto. Si no es así, en el **Explorador de soluciones**, haga doble clic en **MathLibrary.cpp** en la carpeta **Archivos de código fuente** del proyecto **MathLibrary** para abrirlo.
3. En el editor, reemplace el contenido del archivo *Northwind.cs* por el código siguiente:

C++ Copiar

```
// MathLibrary.cpp : Defines the exported functions for the DLL.
#include "pch.h" // use stdafx.h in Visual Studio 2017 and earlier
#include <utility>
#include <limits.h>
#include "MathLibrary.h"

// DLL internal state variables:
static unsigned long long previous_; // Previous value, if any
static unsigned long long current_; // Current sequence value
static unsigned index_; // Current seq. position

// Initialize a Fibonacci relation sequence
// such that F(0) = a, F(1) = b.
// This function must be called before any other function.
void fibonacci_init(
    const unsigned long long a,
    const unsigned long long b)
{
    index_ = 0;
    current_ = a;
    previous_ = b; // see special case when initialized
}

// Produce the next value in the sequence.
// Returns true on success, false on overflow.
bool fibonacci_next()
{
    // check to see if we'd overflow result or position
    if ((ULONG_MAX - previous_ < current_) ||
        (UINT_MAX == index_))
    {
        return false;
    }

    // Special case when index == 0, just return b value
    if (index_ > 0)
    {
        // otherwise, calculate next sequence value
    }
}
```




```
        previous_ += current_;
    }
    std::swap(current_, previous_);
    ++index_;
    return true;
}

// Get the current value in the sequence.
unsigned long long fibonacci_current()
{
    return current_;
}

// Get the current index position in the sequence.
unsigned fibonacci_index()
{
    return index_;
}
```

Para comprobar que todo funciona, compile la biblioteca de vínculos dinámicos. En la barra de menús, elija **Compilar > Compilar solución**. El archivo DLL y la salida del compilador relacionada se colocan en una carpeta llamada *Debug* directamente debajo de la carpeta de la solución. Si crea una compilación de versión, la salida se coloca en una carpeta llamada *Release*. El resultado debe parecerse a este:

Output	 Copiar
<pre>1&gt;----- Build started: Project: MathLibrary, Configuration: Debug Win32 --- --- 1&gt;pch.cpp 1&gt;dllmain.cpp 1&gt;MathLibrary.cpp 1&gt;Generating Code... 1&gt;  Creating library C:\Users\username\Source\Repos\MathLibrary\Debug\MathLibrary.lib and object C:\Users\username\Source\Repos\MathLibrary\Debug\MathLibrary.exp 1&gt;MathLibrary.vcxproj -&gt; C:\Users\username\Source\Repos\MathLibrary\Debug\MathLibrary.dll ===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====</pre>	

Enhorabuena, ha creado un archivo DLL mediante Visual Studio. A continuación, creará una aplicación cliente que usa las funciones exportadas por el archivo DLL.

## Creación de una aplicación cliente que utiliza el archivo DLL

Cuando cree un archivo DLL, piense en cómo pueden usarlo las aplicaciones cliente. Para llamar a las funciones o acceder a los datos exportados por un archivo DLL, el código fuente del cliente debe tener las declaraciones disponibles en tiempo de compilación. En tiempo de vínculo, el enlazador requiere información para resolver las llamadas de función o los accesos a datos. Un archivo DLL proporciona esta información en una *biblioteca de importación*, un archivo que contiene información sobre cómo buscar las funciones y los datos, en lugar del código real. Y en tiempo de ejecución, el archivo DLL debe estar disponible para el cliente, en una ubicación que el sistema operativo puede encontrarlo.

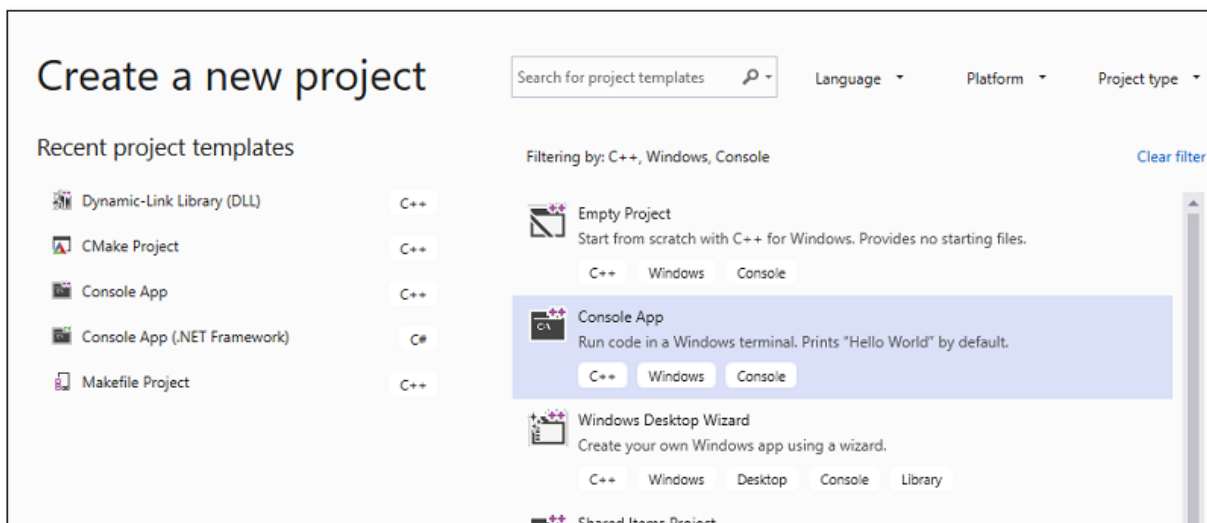
Ya sea propio o de terceros, el proyecto de aplicación cliente necesita varios fragmentos de información para usar un archivo DLL. Debe buscar los encabezados que declaran las exportaciones de DLL, las bibliotecas de importación para el enlazador y el propio archivo DLL. Una solución consiste en copiar todos estos archivos en el proyecto de cliente. Para archivos DLL de terceros que no es probable que cambien mientras el cliente los está desarrollando, este método puede ser la mejor manera de usarlos. Sin embargo, cuando se compila también el archivo DLL, es mejor evitar la duplicación. Si realiza una copia local de los archivos DLL que están en desarrollo, accidentalmente podría cambiar un archivo de encabezado de una copia pero no de otra, o bien usar una biblioteca obsoleta.

Para evitar código sin sincronizar, se recomienda establecer la ruta de inclusión en el proyecto de cliente para que incluya los archivos de encabezado DLL directamente desde el proyecto DLL. Además, establezca la ruta de acceso de la biblioteca en el proyecto de cliente para incluir las bibliotecas de importación de DLL desde el proyecto DLL. Y, por último, copie el archivo DLL compilado desde el proyecto DLL en el directorio de salida de compilación cliente. Este paso permite que la aplicación cliente utilice el mismo código DLL que se compila.

## Para crear una aplicación cliente en Visual Studio

1. En la barra de menús, seleccione **Archivo > Nuevo > Proyecto** para abrir el cuadro de diálogo **Crear un nuevo proyecto**.
2. En la parte superior del cuadro de diálogo, establezca **Lenguaje** en **C++**, establezca **Plataforma** en **Windows** y establezca **Tipo de proyecto** en **Consola**.
3. En la lista de plantillas de proyecto, seleccione **Aplicación de consola** y **Siguiente**.
4. En la página **Configurar el nuevo proyecto**, escriba *MathClient* en el cuadro **Nombre del proyecto** para especificar un nombre para el proyecto. Mantenga los valores predeterminados de **Ubicación** y **Nombre de la solución**. Establezca **Solución** en **Crear**

nueva solución. Desactive **Colocar la solución y el proyecto en el mismo directorio** si está activada.



5. Haga clic en el botón **Crear** para crear el proyecto de cliente.

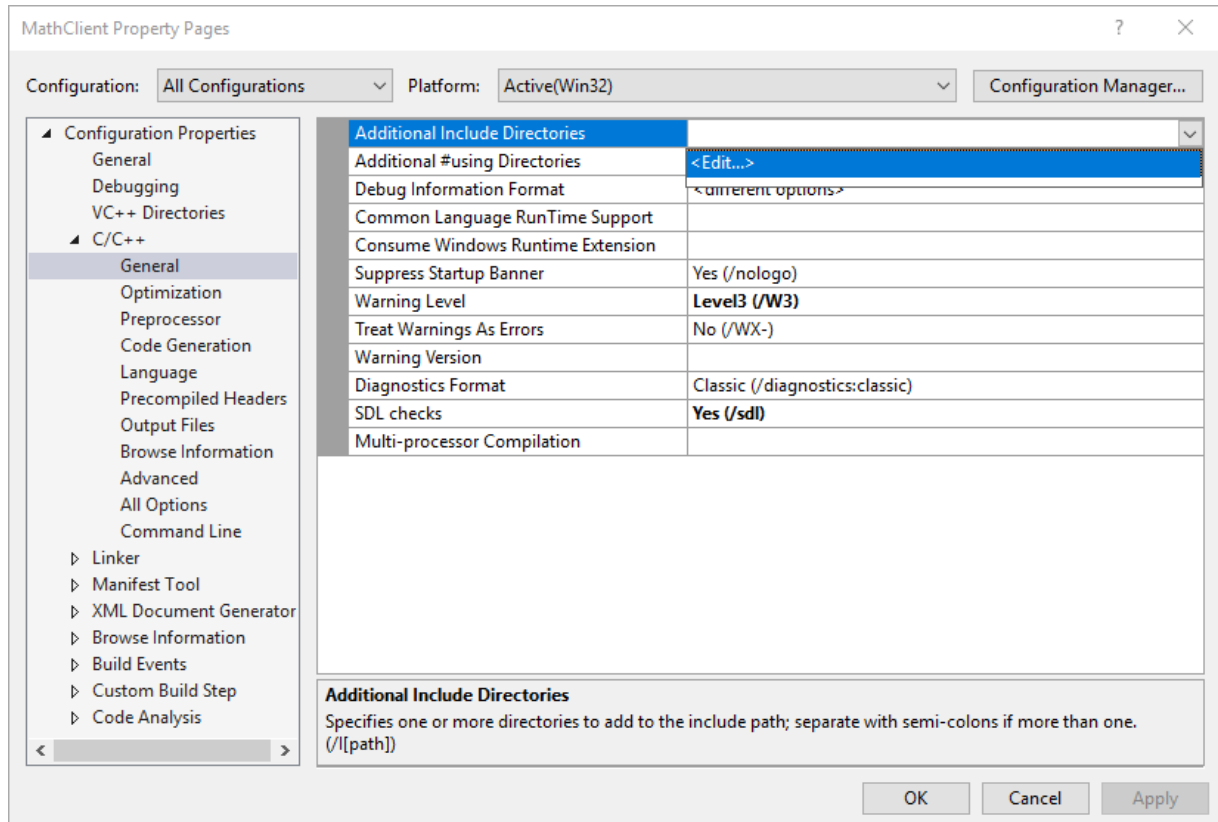
Se crea un proyecto de aplicación de consola mínimo de forma automática. El nombre del archivo de origen principal será el mismo que el elegido anteriormente. En este ejemplo, se denomina **MathClient.cpp**. Se puede compilar, pero aún no usa el archivo DLL.

Después, para llamar a las funciones de MathLibrary en el código fuente, el proyecto debe incluir el archivo *MathLibrary.h*. Puede copiar este archivo de encabezado en el proyecto de aplicación cliente y, a continuación, agregarlo al proyecto como un elemento existente. Este método puede ser una buena elección para las bibliotecas de terceros. Pero si trabaja en el código del archivo DLL y el cliente al mismo tiempo, los archivos de encabezado podrían no estar sincronizados. Para evitar este problema, establezca la ruta de acceso **Directorios de inclusión adicionales** del proyecto para incluir la ruta de acceso en el encabezado original.

## Agregar el encabezado DLL a la ruta de inclusión

1. Haga clic en el botón derecho en el nodo **MathClient** del **Explorador de soluciones** para abrir el cuadro de diálogo **Páginas de propiedades**.
2. En el cuadro de lista desplegable **Configuración**, seleccione **Todas las configuraciones** si no está seleccionado.
3. En el panel de la izquierda, seleccione **Propiedades de configuración > C/C++ > General**.

4. En el panel de propiedades, seleccione el control de lista desplegable junto al cuadro de edición **Directorios de inclusión adicionales** y, a continuación, elija **Editar**.



5. Haga doble clic en el panel superior del cuadro de diálogo **Directorios de inclusión adicionales** para habilitar un control de edición. O bien, elija el icono de carpeta para crear una entrada.
6. En el control de edición, especifique la ruta de acceso a la ubicación del archivo **MathLibrary.h**. Puede elegir el control de puntos suspensivos ( ... ) para buscar la carpeta correcta.

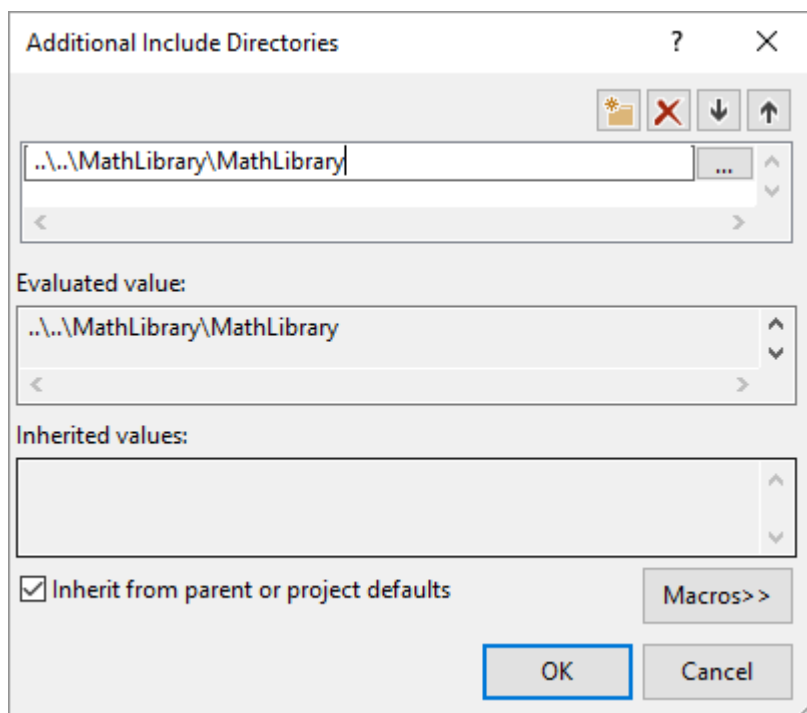
También puede especificar una ruta de acceso relativa de los archivos de origen de cliente en la carpeta que contiene los archivos de encabezado de la DLL. Si ha seguido las indicaciones para colocar el proyecto de cliente en una solución independiente a la DLL, la ruta de acceso relativa debe tener este aspecto:

```
..\..\MathLibrary\MathLibrary
```

Si el archivo DLL y los proyectos de cliente están en la misma solución, la ruta de acceso relativa podría ser similar a la siguiente:


```
..\MathLibrary
```

Cuando el archivo DLL y los proyectos de cliente están en otras carpetas, ajuste la ruta de acceso relativa para que coincida. O bien, use el control de puntos suspensivos para buscar la carpeta.



7. Después de escribir la ruta de acceso al archivo de encabezado en el cuadro de diálogo **Directorios de inclusión adicionales**, seleccione el botón **Aceptar**. En el cuadro de diálogo **Páginas de propiedades**, elija el botón **Aceptar** para guardar los cambios.

Ahora puede incluir el archivo **MathLibrary.h** y utilizar las funciones que declara en la aplicación cliente. Reemplace el contenido de **MathClient.cpp** mediante este código:

C++	 Copiar
<pre>// MathClient.cpp : Client app for MathLibrary DLL. // #include "pch.h" Uncomment for Visual Studio 2017 and earlier #include &lt;iostream&gt; #include "MathLibrary.h"  int main() {     // Initialize a Fibonacci relation sequence.     fibonacci_init(1, 1);     // Write out the sequence values until overflow.     do {         std::cout &lt;&lt; fibonacci_index() &lt;&lt; ": "             &lt;&lt; fibonacci_current() &lt;&lt; std::endl;     } while (fibonacci_next());     // Report count of values written before overflow.     std::cout &lt;&lt; fibonacci_index() + 1 &lt;&lt;         " Fibonacci sequence values fit in an " &lt;&lt;</pre>	

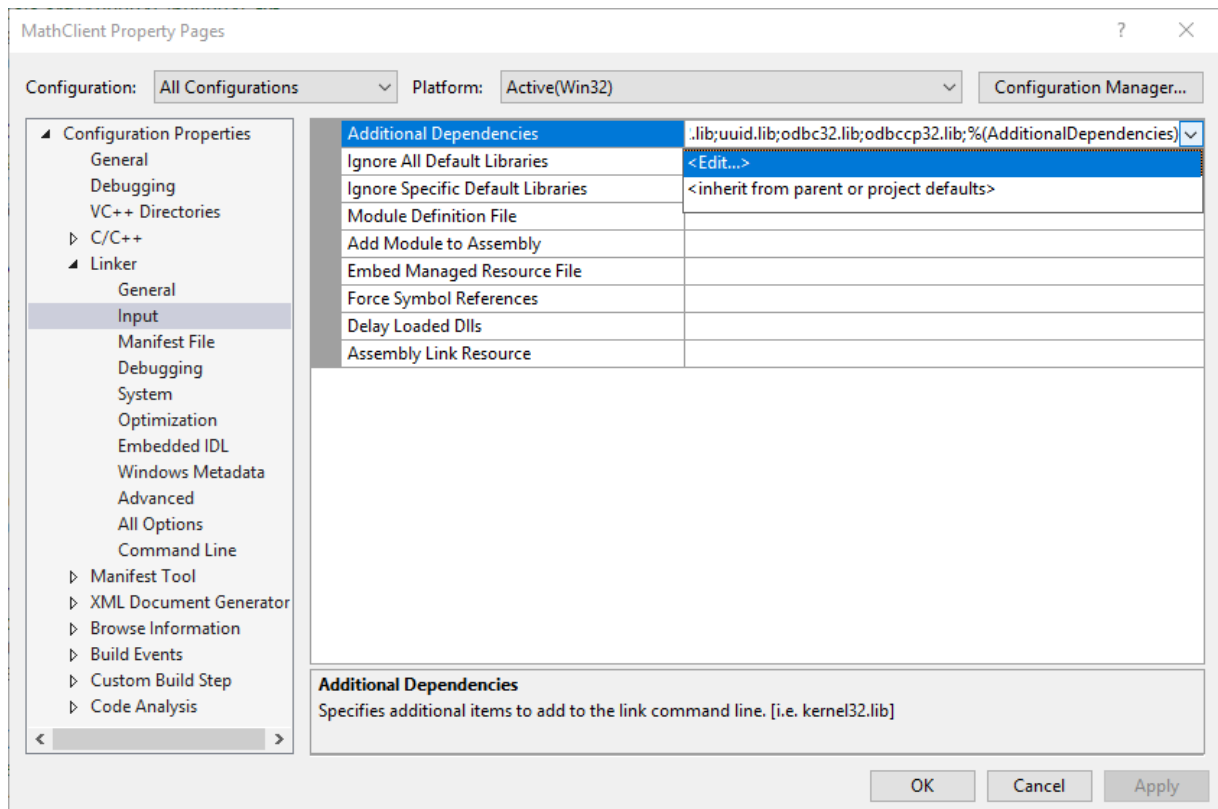
```
"unsigned 64-bit integer." << std::endl;  
}
```

Este código se puede compilar, pero no vincular. Si compila la aplicación cliente ahora, la lista de errores muestra varios errores LNK2019. Esto se debe a que falta información en el proyecto: Todavía no ha especificado que el proyecto tiene una dependencia de la biblioteca *MathLibrary.lib*. Además, no ha indicado al enlazador cómo encontrar el archivo *MathLibrary.lib*.

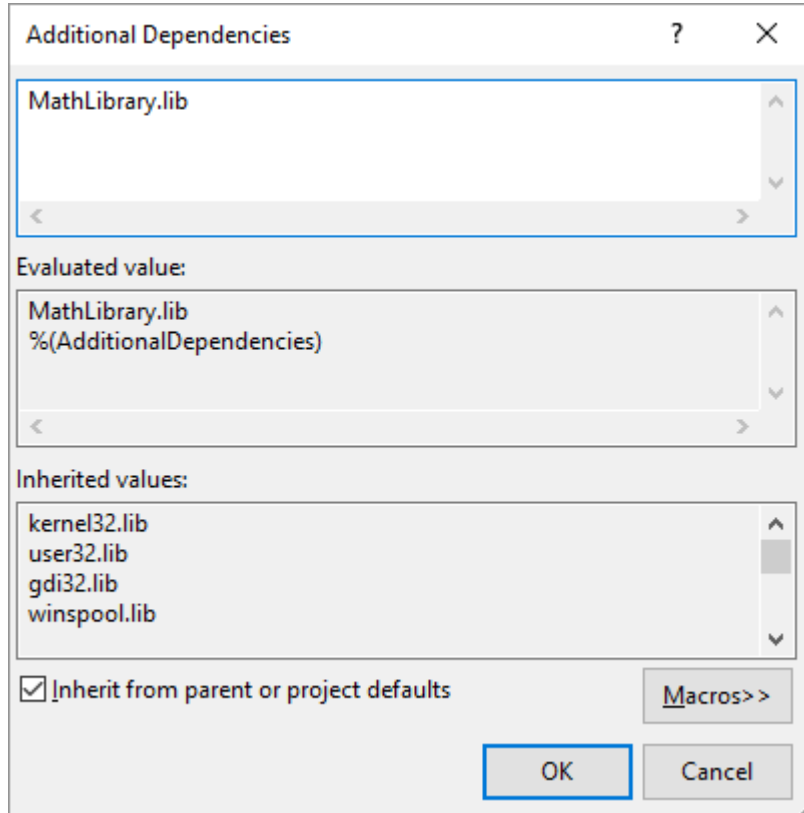
Para corregir este problema, puede copiar el archivo de biblioteca directamente en el proyecto de aplicación cliente. El enlazador lo encontrará y lo usará de forma automática. Pero si la biblioteca y la aplicación cliente están en desarrollo, eso podría dar lugar a cambios en una copia que no se muestran en la otra. Para evitar este problema, puede establecer la propiedad **Dependencias adicionales** para indicar al sistema de compilación que el proyecto depende de *MathLibrary.lib*. Y puede establecer una ruta de acceso **Directorios de bibliotecas adicionales** en el proyecto para incluir la ruta de acceso a la biblioteca original al vincular.

## Agregar la biblioteca de importación de DLL al proyecto

1. Haga clic con el botón derecho en el nodo **MathClient** del **Explorador de soluciones** y seleccione **Propiedades** para abrir el cuadro de diálogo **Páginas de propiedades**.
2. En el cuadro de lista desplegable **Configuración**, seleccione **Todas las configuraciones** si no está seleccionado. Garantiza que los cambios de propiedad se aplican a las compilaciones de depuración y versión.
3. En el panel de la izquierda, seleccione **Propiedades de configuración > Enlazador > Entrada**. En el panel de propiedades, seleccione el control de lista desplegable junto al cuadro de edición **Dependencias adicionales** y, a continuación, elija **Editar**.

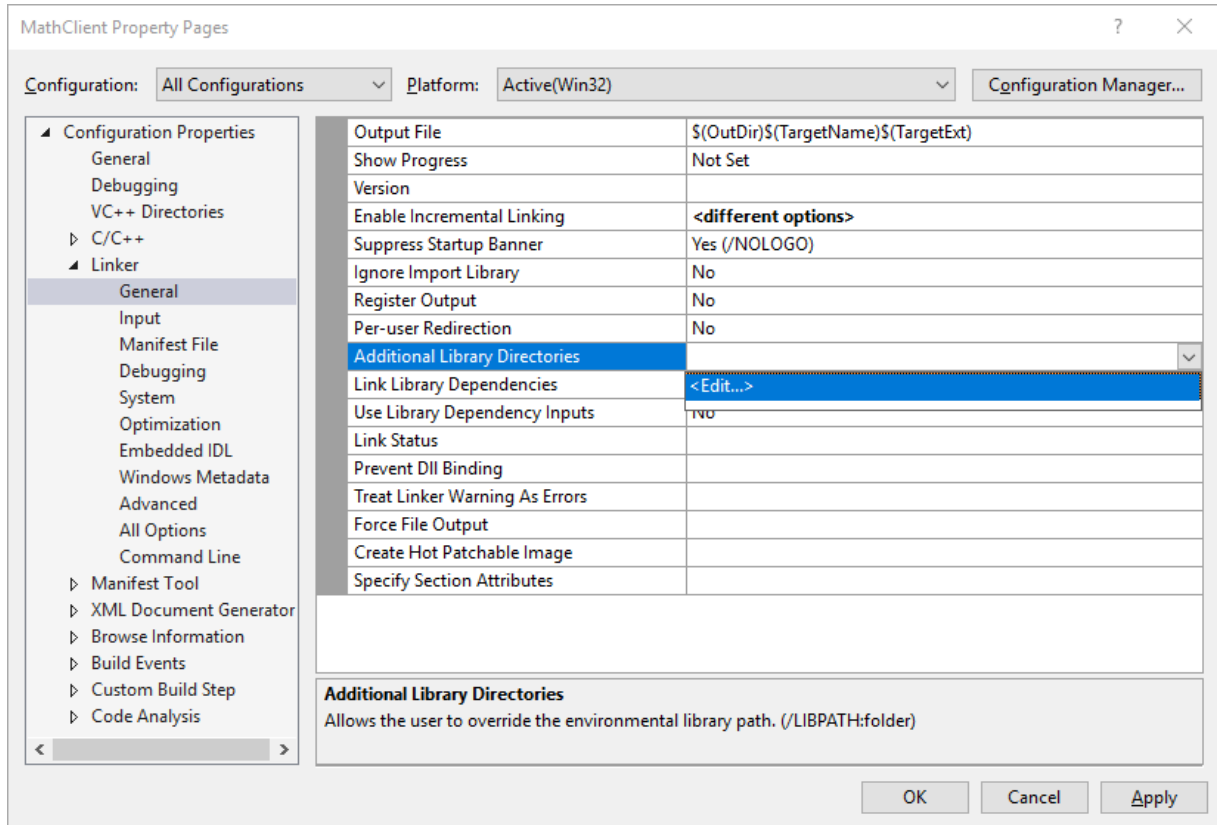


4. En el cuadro de diálogo **Dependencias adicionales**, agregue *MathLibrary.lib* a la lista del control de edición superior.



5. Elija **Aceptar** para volver al cuadro de diálogo **Páginas de propiedades**.

6. En el panel de la izquierda, seleccione **Propiedades de configuración > Enlazador > General**. En el panel de propiedades, seleccione el control de lista desplegable junto al cuadro de edición **Directorios de bibliotecas adicionales** y, a continuación, elija **Editar**.

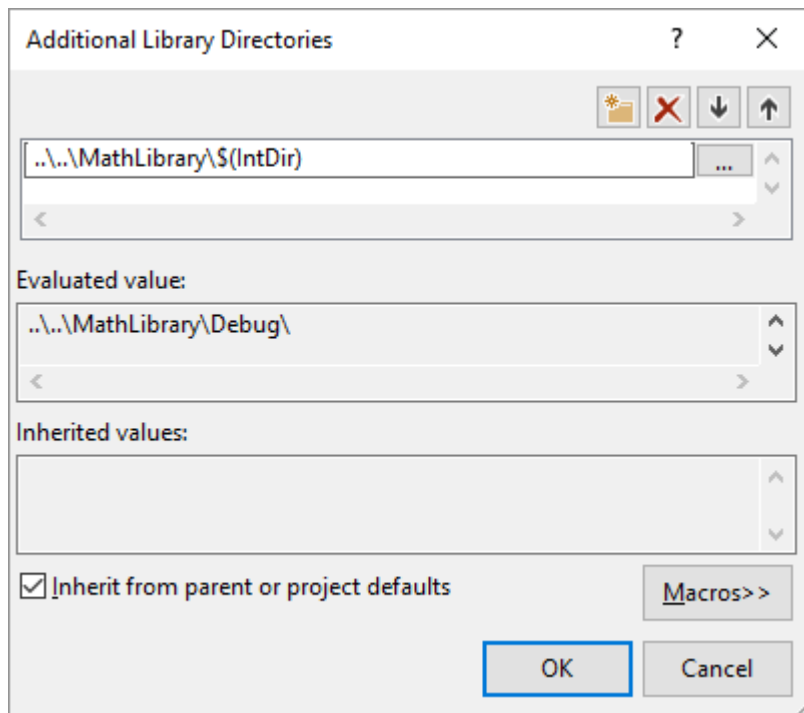


7. Haga doble clic en el panel superior del cuadro de diálogo **Directorios de bibliotecas adicionales** para habilitar un control de edición. En el control de edición, especifique la ruta de acceso a la ubicación del archivo **MathLibrary.lib**. De forma predeterminada, se encuentra en una carpeta llamada *Debug* directamente en la carpeta de la solución DLL. Si crea una compilación de versión, el archivo se coloca en una carpeta llamada *Release*. Puede usar la macro `$(IntDir)` para que el enlazador pueda encontrar el archivo DLL, con independencia del tipo de compilación que cree. Si ha seguido las indicaciones para colocar el proyecto de cliente en una solución independiente al proyecto DLL, la ruta de acceso relativa debe tener este aspecto:

```
..\..\MathLibrary\$(IntDir)
```

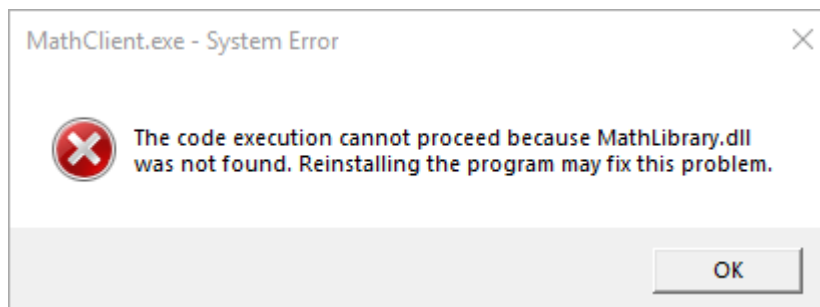
Si el archivo DLL y los proyectos de cliente están en otras ubicaciones, ajuste la ruta de acceso relativa para que coincida.





8. Después de escribir la ruta de acceso al archivo de biblioteca en el cuadro de diálogo **Directorios de bibliotecas adicionales**, elija el botón **Aceptar** para volver al cuadro de diálogo **Páginas de propiedades**. Seleccione **Aceptar** para guardar los cambios de propiedad.

La aplicación cliente ahora se puede compilar y vincular correctamente, pero todavía no tiene todo lo que necesita para ejecutarse. Cuando el sistema operativo carga la aplicación, busca el archivo DLL MathLibrary. Si no encuentra el archivo DLL en ciertos directorios del sistema, la ruta de acceso de entorno o el directorio de aplicaciones local, se produce un error de carga. En función del sistema operativo, verá un mensaje de error similar al siguiente:



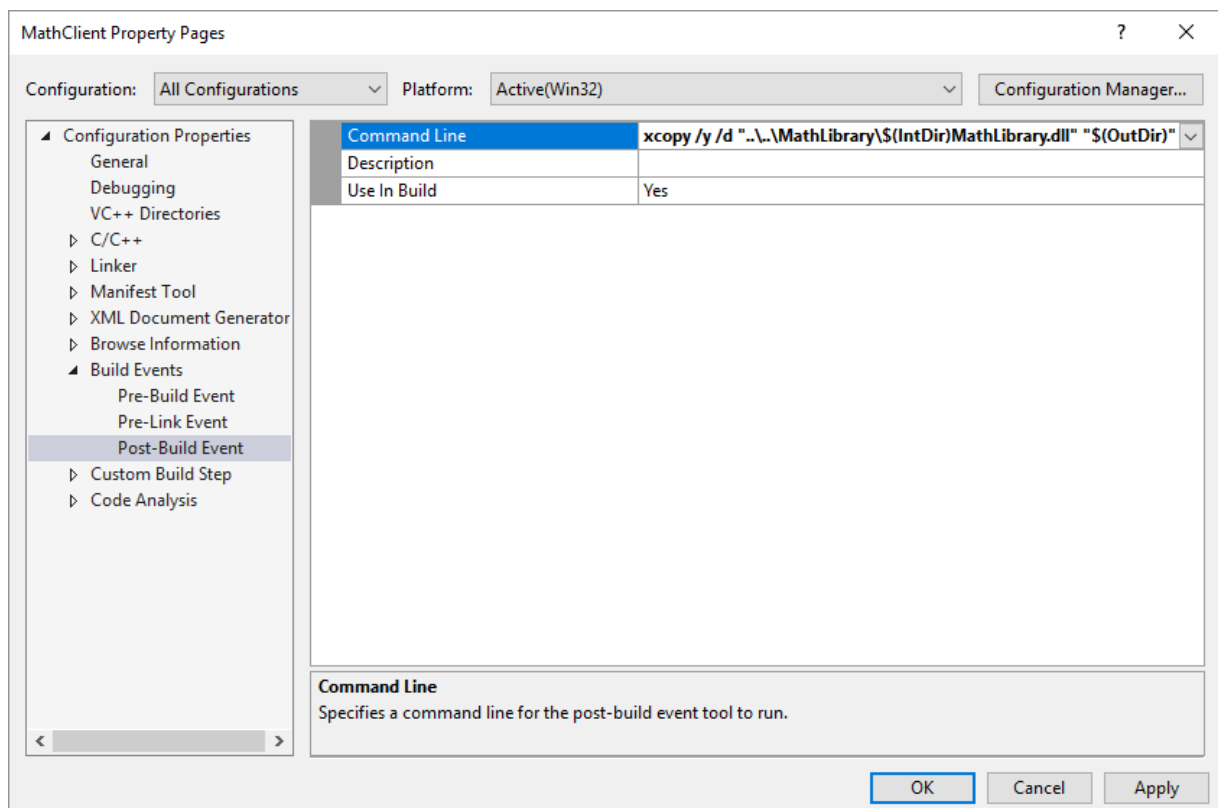
Una forma de evitar este problema es copiar el archivo DLL en el directorio que contiene el archivo ejecutable del cliente como parte del proceso de compilación. Puede agregar un **evento posterior a la compilación** al proyecto, para agregar un comando que copie el archivo DLL en el directorio de salida de compilación. El comando especificado aquí solo copia el archivo DLL si falta o ha cambiado. Usa macros para copiar a y desde las ubicaciones de depuración o de versión, en función de la configuración de compilación.

# Copiar el archivo DLL en un evento posterior a la compilación

1. Haga clic con el botón derecho en el nodo **MathClient** del **Explorador de soluciones** y seleccione **Propiedades** para abrir el cuadro de diálogo **Páginas de propiedades**.
2. En el cuadro desplegable **Configuración**, seleccione **Todas las configuraciones** si aún no se ha seleccionado.
3. En el panel de la izquierda, seleccione **Propiedades de configuración > Eventos de compilación > Evento posterior a la compilación**.
4. En el panel de propiedades, seleccione el control de edición en el campo **Línea de comandos**. Si ha seguido las indicaciones para colocar el proyecto de cliente en una solución independiente al proyecto DLL, escriba este comando:

```
xcopy /y /d "..\..\MathLibrary\$(IntDir)MathLibrary.dll" "$(OutDir)"
```

Si el archivo DLL y los proyectos de cliente están en otros directorios, ajuste la ruta de acceso relativa a la DLL para que coincida.

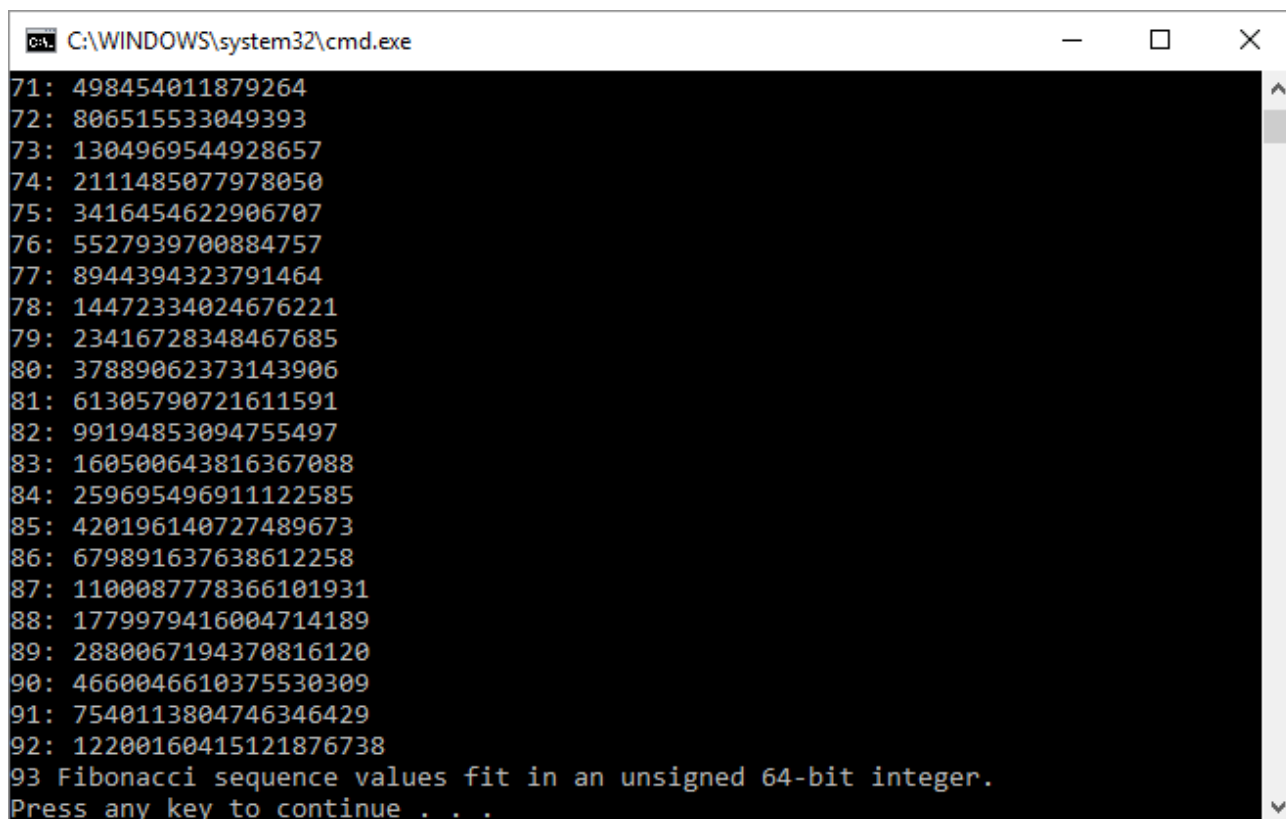


5. Elija el botón **Aceptar** para guardar los cambios en las propiedades del proyecto.

Ahora la aplicación cliente tiene todo que lo necesario para compilarse y ejecutarse. Compile la aplicación seleccionando **Compilar > Compilar solución** en la barra de menús. En la ventana **Salida** de Visual Studio se debe mostrar algo parecido al ejemplo siguiente, en función de la versión de Visual Studio:

Output	Copiar
<pre>1&gt;----- Build started: Project: MathClient, Configuration: Debug Win32 ---- -- 1&gt;MathClient.cpp 1&gt;MathClient.vcxproj -&gt; C:\Users\username\Source\Repos\MathClient\Debug\MathClient.exe 1&gt;1 File(s) copied ===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====</pre>	

Ha creado una aplicación que llama a funciones del archivo DLL. Ahora ejecute la aplicación para ver lo que hace. En la barra de menús, seleccione **Depurar > Iniciar sin depuración**. Visual Studio abre una ventana de comandos para que se ejecute el programa. La última parte de la salida debe tener un aspecto parecido al siguiente:



```
C:\WINDOWS\system32\cmd.exe
71: 498454011879264
72: 806515533049393
73: 1304969544928657
74: 2111485077978050
75: 3416454622906707
76: 5527939700884757
77: 8944394323791464
78: 14472334024676221
79: 23416728348467685
80: 37889062373143906
81: 61305790721611591
82: 99194853094755497
83: 160500643816367088
84: 259695496911122585
85: 420196140727489673
86: 679891637638612258
87: 1100087778366101931
88: 1779979416004714189
89: 2880067194370816120
90: 4660046610375530309
91: 7540113804746346429
92: 12200160415121876738
93 Fibonacci sequence values fit in an unsigned 64-bit integer.
Press any key to continue . . .
```

Pulse cualquier tecla para cerrar la ventana de comandos.

Ahora que ha creado un archivo DLL y una aplicación cliente, puede experimentar. Intente establecer puntos de interrupción en el código de la aplicación cliente y ejecutar la aplicación

en el depurador. Vea qué sucede cuando depure paso a paso por instrucciones una llamada a la biblioteca. Agregue otras funciones a la biblioteca o escriba otra aplicación cliente que use el archivo DLL.

Al implementar la aplicación, también debe implementar los archivos DLL que utiliza. La manera más sencilla para que los archivos DLL que compile o que incluya de terceros estén disponibles es colocarlos en el mismo directorio que la aplicación. Se conoce como *implementación local de la aplicación*. Para obtener más información sobre la implementación, vea [Deployment in Visual C++](#).

## Vea también

[Llamada a funciones de un archivo DLL desde aplicaciones programadas en Visual Basic](#)

---

¿Le ha resultado útil esta página?

 Yes  No

---