<style>@import url(https://fonts.googleapis.com/css?kit=p2CikV5640FLmGAyoASwDnb_WvOJImzoegEymcA3TWY);</style>

Tutorial para compilar usando librerias externas en C++

por Javi Agenjo 2010

¿Qué es una libreria?

- Es un conjunto de funciones diseñado para facilitar una tarea específica y ahorrar trabajo de programación.
- Las librerias suelen estar pensadas para ser utilizadas en un lenguaje concreto (aunque a veces tienen versiones para diferentes lenguajes)
- Pueden venir como código fuente o ya compiladas listas para ser usadas.

- Todo lenguaje de programación suele venir con unas librerias básicas (normalmente conocidas como librerias estandar).
- Podemos utilizar cuantas librerias queramos al desarrollar una aplicación.

¿Para qué sirven las librerias?

- Nos ayudan a reducir la cantidad de código que tenemos que desarrollar (no merece la pena reinventar la rueda cada vez).
- Nos permiten acceder a funcionalidades del sistema que sino serian demasiado complejas de utilizar (GPU, sockets, threads, etc).
- Reducen el numero de bugs (fallos en el codigo) ya que las librerias suelen estar desarrolladas y testeadas por profesionales.
- Gracias a las librerias resulta fácil acceder desde tu código a funcionalidades como la red, el sistema operativo, el hardware, o conceptos más de algoritmos.
- Podemos crear nuestras propias librerias para reusar código entre diferentes proyectos.

Donde conseguir librerias?

Una vez sabemos lo que queremos desarrollar podemos buscar en internet donde encontraremos muchas librerias

diferentes para cada tarea, debemos elegir la que mejor se adhiera a nuestras necesidades.

Normalmente bajaremos la libreria que mejor se adapte a nuestro entorno de desarrollo (Visual Studio, GCC, etc) y a nuestra arquitectura (Windows, 64bits, x86, OSX, Unix).

Una vez hemos bajado la librería debemos configurar tanto el compilador como nuestro proyecto para que las utilice o de lo contrario no sabrá donde encontrarla.

Cómo funciona una libreria?

Durante la fase de compilado el compilador traduce el codigo fuente nuestro a lenguaje del sistema, creando los archivos objeto, pero tan solo se limita a evaluar que la sintaxis sea correcta y a efectuar la traducción. No ensambla las diferentes funciones entre ellas para crear el ejecutable, esa parte la realiza el enlazador (o vinculador o linker).

Si tenemos el código fuente de la libreria bastará compilar el codigo fuente suyo junto con el codigo de nuestra aplicación cuando queramos generar el ejecutable.

Si tenemos la libreria ya compilada entonces tenemos que decirle al linker que cuando busque funciones lo haga tambien dentro de los archivos objeto de la libreria.

Dependerá de la libreria que usemos, se recomienda usar librerias ya compiladas ya que reducen el tiempo a la hora de compilar.

Como trabaja el compilador y el linker? (1/2)

- Evaluan que todas las llamadas a funciones tengan la sintaxis correcta (para esto hace uso de los headers de las funciones, donde se especifica el nombre de la funcion, el numero de parametros y el tipo de estos).
- Si una función tiene algun error de sintaxis el compilador genera un error de compilado indicando la linea del error.
- Si todas las llamadas siguen las reglas entonces genera los archivos objeto con todas las funciones compiladas.
- El compilador NO verifica que las funciones a las que llama cada functión existan, es decir, si yo llamo desde mi código a una función llamada getDate el compilador solo mira que exista un header donde se defina tal función (para saber el formato), pero no comprueba si tiene cuerpo (si alguien la ha programado).
- Si todo es correcto genera los archivos intermedios.

Como trabaja el compilador y el linker? (2/2)

- El linker coge los archivos intermedios (el codigo compilado de cada función) y los enlaza entre ellos acorde a las llamadas que hace cada función, para generar el ejecutable final.
- Para cada llamada a una función el linker busca entre todos los archivos objeto especificados funciones con ese nombre y esos parametros.
- Si usamos alguna librería externa DEBEMOS decirle al linker que la use, sino no encontrará las funciones.
- Si una función llama a otra función pero el linker no encuentra en los archivos objeto ninguna que coincida entonces genera un error de 'linkado'. El error NO indicará ninguna linea ya que el linker no tiene el código original.

- En el error podremos ver qué función no encuentra y quien la llamaba.

Tipos de librerias

Existen dos tipos de librerias:

- Librerias estáticas: son aquellas pensadas para que se integren dentro del ejecutable, es decir, el codigo de las funciones de la libreria irá incluido en el archivo junto con el código de las funciones que tú has programado. Se resuelve al compilar la aplicación. En Windows las librerias estáticas suelen tener extension .LIB
- Librerias dinámicas: son librerias en las que el código compilador va a parte y cuando el archivo ejecutable las necesita se las pide al sistema operativo, este busca el archivo, y si lo encuentra busca dentro la función que necesita y se la pasa al ejecutable. Aquí se resuelve en tiempo de ejecución, puede que compile correctamente pero al ejecutarla nos de un error. En Windows las librerias dinamicas suelen tener extensión .DLL (dynamic link library)

¿Por qué se usan librerias dinámicas?

Una vez vistas las librerias estáticas da la impresión de que no tiene mucho sentido usar librerias dinamicas, ya que complican más el proceso y añaden un posible error extra (que el archivo con la libreria dinamica no esté junto con el ejecutable y el sistema no lo encuentre).

Existen varios motivos para usarlas:

- A veces queremos que parte del código de la aplicación pueda ser modificado sin necesidad de recompilar cada vez la aplicación.

- A veces nuestra aplicación necesita usar funciones que no hemos desarrollado nosotros y del que no tenemos el código fuente.

Un ejemplo de esto son los drivers que estan almacenados en librerias dinámicas.

El sistema operativo accede a funciones que usan el hardware y emplea el driver como intermediario, si quiero puedo actualizar el driver cambiando la librería que lo contiene, y no por eso tengo que recompilar todas las aplicaciones que lo usan.

Qué estructura tienen las librerias?

Cuando te bajas una libreria de internet suelen venir con la siguiente estructura:

- Headers (carpeta include): es donde estan las cabeceras de las funciones, le sirve al compilador para saber si las llamadas a las funciones de la libreria son correctas. Se incluyen poniendo #include "nombre_header.h"
- Libs (carpeta libs): es donde estan los archivos objeto de la libreria, el codigo ya compilado. Hay que especificarle al linker que use estos archivos cuando usemos la libreria o dará errores de linkado.
- Binarios (carpeta bin): es donde estan las librerias dinamicas, ya que algunas librerias ponen parte del codigo en librerias estáticas y parte en librerias dinamicas.
- Documentacion (carpeta doc): informacion sobre cómo usar la librería, parametros, ejemplos, etc.

Algunas librerias pueden incluir además el código fuente y proyectos para compilarlo por uno mismo

Cómo instalo una libreria?

Normalmente cada entorno de desarrollo tiene diferentes

carpetas donde se guardan las librerias estandar que utiliza, basta copiar en esas carpetas cada cosa.

Los headers en la carpeta de headers y las librerias estaticas en la de librerias estáticas. O podemos indicarle a nuestro compilador las carpetas donde tiene que acceder a buscar las librerias que no encuentre.

Sin embargo las librerias dinámicas tienen que copiarse en una carpeta en la que el sistema operativo busque por defecto cuando se le pide un archivo, estas carpetas son las que estan especificadas en el PATH del sistema.

En windows por ejemplo se usa c:/windows/system32

Cómo le digo al linker en qué archivos tiene que buscar las funciones compiladas

Depende del compilador y del IDE que uses. Un entorno de desarrollo (IDE) es un programa pensado para facilitar la tarea de programación.

- Si es Visual Studio tienes que configurar el proyecto y el entorno de desarrollo adecuadamente (ver siguientes diapositivas).
- Si el compilador es GCC hay que editar el Makefile adecuadamente para que el compilador reciba por parametro las librerias o indicarselo por parametro.
- Los IDEs de OSX se basan en GCC aunque no se vea a simple vista, para ello tienes que usar diferentes menus para indicar los parametros de GCC.

Configurar en Visual Studio 2008 (1/3)

Desde 'Herramientas -> Opciones' configurar el IDE para que sepa en qué carpetas buscar. Para 'Archivos de Inclusión' (headers) y para 'Archivos de bibioteca' (libs)

Configurar en Visual Studio 2008 (2/3)

Ademas de decirle al IDE donde buscar las librerias tenemos que decirle al proyecto nuestro en qué librerias estáticas buscar cuando compile.

Por defecto Visual Studio ya busca en una serie de librerias estáticas que vienen por defecto, como por ejemplo GL.lib o GLUT.lib por lo que no es necesario añadirlas.

Lo mejor es añadir unicamente las que utilicemos, si no vamos a usar SDL no añadiremos SDL.lib

Si queremos añadir alguna libreria estática lo haremos en la siguiente ventana (continua...)

Configurar en Visual Studio 2008 (3/3)

Configurar el proyecto desde `Proyecto -> Propiedades`

Configurar en Visual Studio 2010...12

En las nuevas versiones de Visual Studio han quitado lo de poder configurar las librerias para todos los proyectos, necesitas hacerlo por proyecto en las propiedades del proyecto.:

Configurar librerias en GCC

Si trabajamos desde un sistema operativo basado en UNIX los conceptos son los mismos aunque en la practica hay variaciones.

Una vez instaladas las librerias en el sistema operativo (usando un gestor de paquetes) debemos añadir el parametro -l (menos ele) seguido del nombre de la libreria estática, por ejemplo, linkar con OpenGL (el archivo con la libreria se llama GL)

```
gcc *.cpp -lGL -o ./my_app
```

Aun así los IDEs que usan GCC suelen tener opciones donde poder configurar facilmente estos parametros.

Macros de preprocesador

Suele ser habitual querer usar los mismos archivos de código para compilar sobre diferentes plataformas. Para ello podemos especificarle al compilador que ignore ciertas lineas en función de qué macros (palabras clave) han sido definidas en los settings del proyecto.

Desde código podemos poner lo siguiente:

```
#ifdef MI_MACRO
//specific code
//...
#endif
```

Este código solo se compilaría si en las propiedades del proyecto se ha especificado que se use esa macro.
Existen algunas macros que vienen por defecto dependiendo del compilador, como WIN32, _DEBUG, o _CONSOLE

En Visual Studio estan en C/C++ -> Preprocesador

Errores comunes

fatal error C1083: No se puede abrir el archivo incluir: 'foo': No such file or directory

Incluir una librería a la que el compilador no sabe llegar

Si ponemos `#include "my_library.h"` el compilador buscará ese archivo en la carpeta raiz del proyecto donde está nuestro código, si no la encuentra ahí la buscará en las carpetas que tenga configuradas. Si pese a todo no las encuentra dará un error al compilar.

No podemos esperar que el compilador recorra todo el disco duro buscando el archivo, solo lo buscará donde le digamos,

así que es importante revisar las rutas de los archivos y asegurarse que los copiamos en la carpeta adecuada.

Errores comunes

error LNK2001: símbolo externo "int __cdecl my_function(void)" (?my_function@@YAHXZ) sin resolver

No especificarle al linker en qué archivos buscar el cuerpo de las funciones

El linker puede tener miles de archivos objeto cada uno con cientos de funciones ya compiladas. Cuando usamos una función él busca SOLO en aquellos que se le ha indicado, no podemos esperar que busque en todos los archivos.

Es por eso que tenemos que decirle qué archivos objeto (. libs en windows) tiene que usar.