Explicit Linking

The harder way to load a DLL is a little bit more complicated. You will need function pointers and some Windows functions. But, by loading DLLs this way, you do not need the .lib or the header file for the DLL, only the DLL. I'll list some code and then explain it.

```
#include <iostream>
#include <windows.h>

typedef int (*AddFunc)(int,int);
typedef void (*FunctionFunc)();

int main()
{
   AddFunc _AddFunc;
   FunctionFunc _FunctionFunc;
   HINSTANCE hInstLibrary = LoadLibrary("DLL_Tutorial.dll");

   if (hInstLibrary)
   {
      _AddFunc = (AddFunc)GetProcAddress(hInstLibrary, "Add");
      _FunctionFunc = (FunctionFunc)GetProcAddress(hInstLibrary,
         "Function");

      if (_AddFunc)
      {
         std::cout << "23 = 43 = " << _AddFunc(23, 43) << std::endl;
      }
      if (_FunctionFunc)
      {
         _FunctionFunc();
      }

      FreeLibrary(hInstLibrary);
   }
   else
   {
      std::cout << "DLL Failed To Load!" << std::endl;
   }

   std::cin.get();

   return 0;
}
```

The first thing you'll notice is that you included the file "windows.h" and removed "DLL_Tutorial.h". The reason is simply because Windows.h contains many Windows functions and you will need only a few right now. It also contains some Windows-specific variables that you will use. You can remove the DLL's header file (DLL_Tutorial.h) because, as I've stated before, you don't need it when you load DLLs this way.

The next thing you'll notice is an odd-looking piece of code in the form of:

```
typedef int (*AddFunc)(int,int);
typedef void (*FunctionFunc)();
```

Those are function pointers. Because this is a tutorial about DLLs, an in-depth look at function pointers is out of the scope of this tutorial; so, for now just think of them as aliases for the functions the DLL contains. I like to name them with the word "Func" attached on the end. The (int,int) part is the parameters that the function takes; for example, the Add function takes in two ints; therefore, you need those as the parameters for the function pointer. The Function function takes no parameters, so you can leave that blank. The first two lines in main() are the function pointers being declared so that you can set them equal to the functions inside the DLL. I just like to define them beforehand.

An HINSTANCE is a Windows dat type that is a handle to an instance; in this case, that instance will be the DLL. You get the instance of the DLL by using the LoadLibrary() function; it takes in a name as the parameter. After the call to LoadLibrary, you must check to see whether the function succeeded. You can do so by checking whether the HINSTANCE is equal to NULL (defined as 0 in Windows.h or one of the headers Windows.h includes). If it is equal to NULL, the handle is not valid, and you must free the library. In other words, you must free up the memory that the DLL was taking up. If the function succeeded, your HINSTANCE contains the handle to the DLL.

Once you have the handle to the DLL, you now can retrieve the functions from the DLL. To do that, you must use the GetProcAddress() function, which takes in as parameters the handle to the DLL (you can use the HINSTANCE) and the name of the function. You set the function pointers to contain the value returned by GetProcAddress() and you must cast GetProcAddress() to the function pointer that you defined for that function. For example, for the Add() function, you must cast GetProcAddress() to AddFunc; this is so that it knows the parameters and return type. Now, it would be wise to make sure that the function pointers are not equal to NULL and that they hold the functions of the DLL. That is just a simple if statement; if one of them does equal NULL, you must free the library as mentioned above.

Now that the function pointers hold the functions of the DLL, you can use them, but there is one catch: You cannot use the actual function name; you must use the function pointer to call them. After that, all you need to do is free the library and that's it.

Now you know the basics of DLLs. You know how to create them, and you know how to link them with two different methods. There is still more to learn about this, but I'll leave that to you to look for and for better writers to write.