C++ Static and Dynamic Libraries

Administrative

- Test next Thursday 10/13/16
- **Readings,Lectures,Examples: up to the red cutoff**

Outline

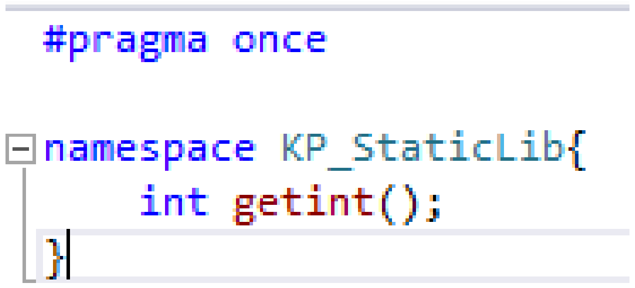
- Why bother?
- Static verses Dynamic Linking

Why Bother?

- How most team based projects are managed
- Given interface and expected behavior
- Produce a library that delivers requirements
- Also reusable
- Excellent encapsulation, I code to the interface your library has, I don't care how its implemented
- Also how most 3rd party software is delivered for programmer use.

Whats an interface?

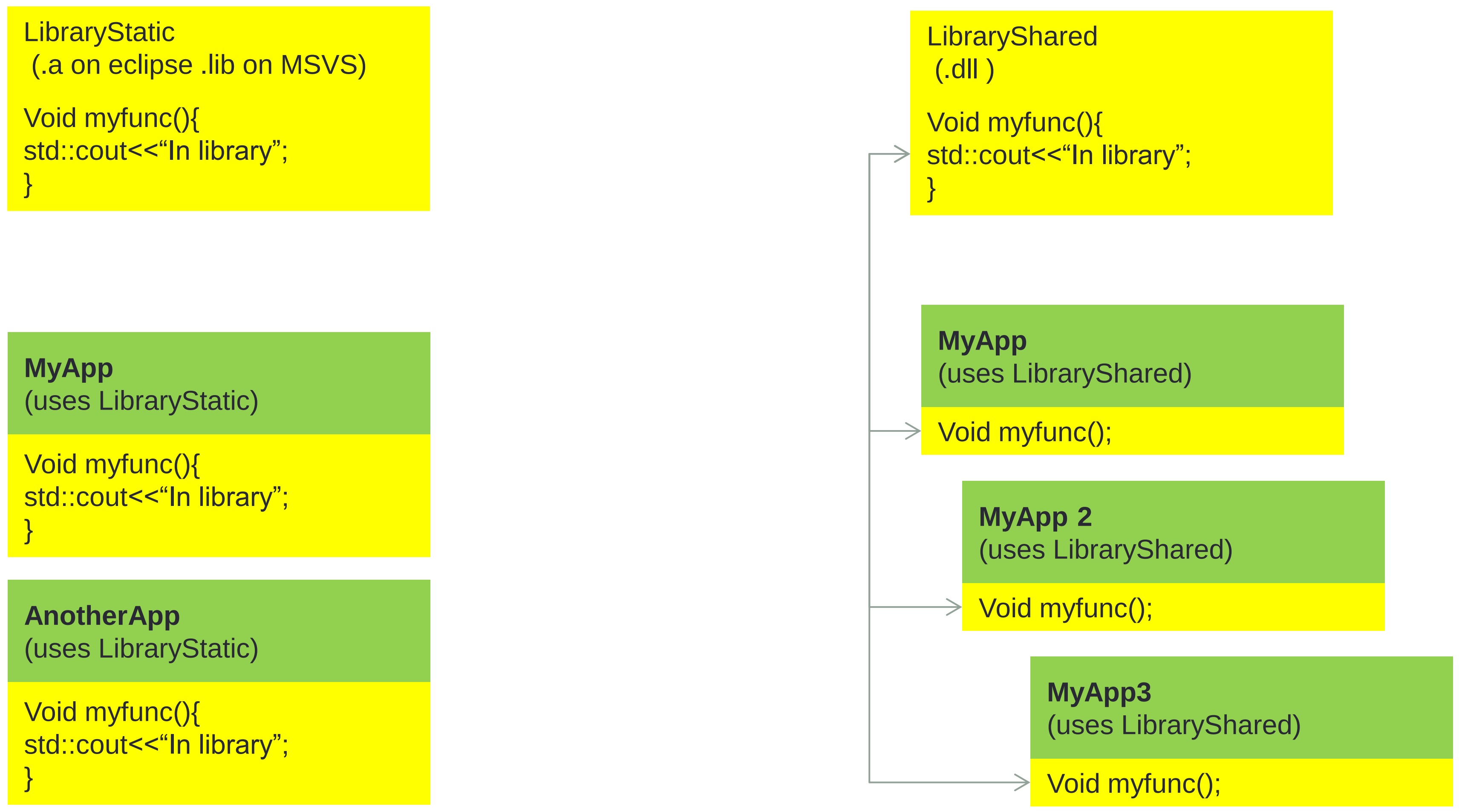
- Public access to your library.
- Typically provided via header files.
- For instance here is the interface to a static library that I will discuss today for the library StaticLibrary;



```
#pragma once

namespace KP_StaticLib{
	int getint();
}
```

Static Verses Dynamic



Static linkage

LibraryStatic
(.a on eclipse .lib on MSVS)

```
Void myfunc(){
std::cout<<“In library”;
}
```

MyApp
(uses LibraryStatic)

```
Void myfunc(){
std::cout<<“In library”;
}
```

AnotherApp
(uses LibraryStatic)

```
Void myfunc(){
std::cout<<“In library”;
}
```

Shared linkage

LibraryShared
(.dll)

```
Void myfunc(){
std::cout<<“In library”;
}
```

MyApp
(uses LibraryShared)

```
Void myfunc();
```

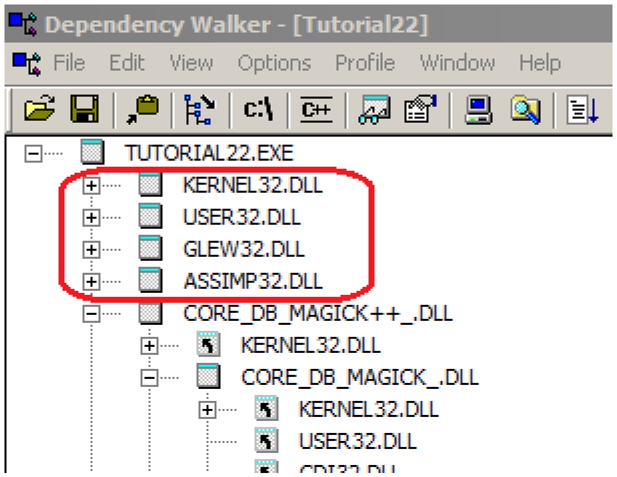
MyApp 2
(uses LibraryShared)

```
Void myfunc();
```

MyApp3
(uses LibraryShared)

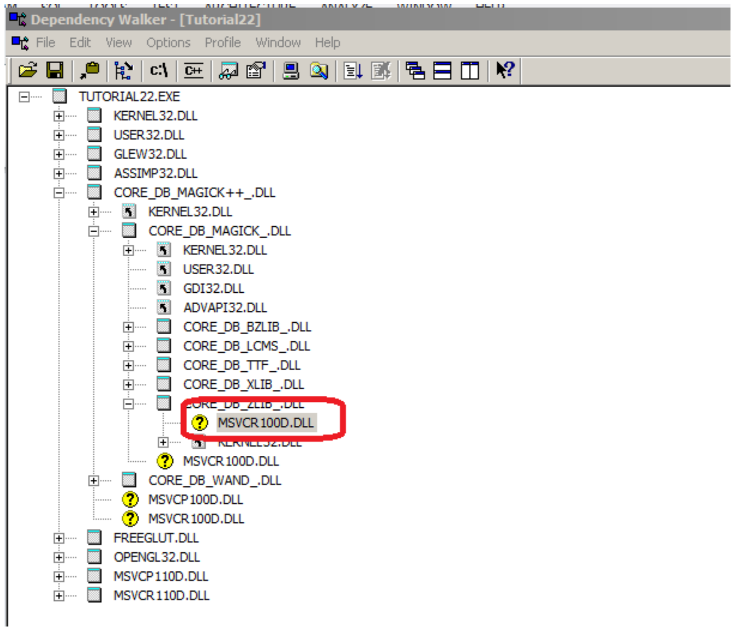
```
Void myfunc();
```

How it appears in Dependency Walker Dynamic Link


Dependency Walker - [Tutorial22]
File Edit View Options Profile Window Help

```
TUTORIAL22.EXE
    KERNEL32.DLL
    USER32.DLL
    GLEW32.DLL
    ASSIMP32.DLL
    CORE_DB_MAGICK++_.DLL
        KERNEL32.DLL
        CORE_DB_MAGICK_.DLL
            KERNEL32.DLL
            USER32.DLL
```

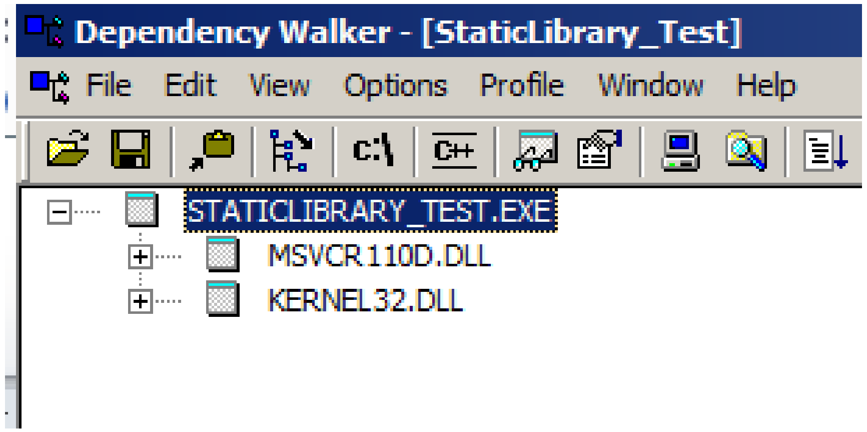
DLL – When things go wrong


Dependency Walker - [Tutorial22]
File Edit View Options Profile Window Help

```
TUTORIAL22.EXE
    KERNEL32.DLL
    USER32.DLL
    GLEW32.DLL
    ASSIMP32.DLL
    CORE_DB_MAGICK++_.DLL
        KERNEL32.DLL
        CORE_DB_MAGICK_.DLL
            KERNEL32.DLL
            USER32.DLL
            GDI32.DLL
            ADVAPI32.DLL
            CORE_DB_BZLIB_.DLL
            CORE_DB_LCMS_.DLL
            CORE_DB_TTF_.DLL
            CORE_DB_XLIB_.DLL
            CORE_DB_ZLIB_.DLL
                MSVCR100D.DLL
                KERNEL32.DLL
            MSVCR100D.DLL
        CORE_DB_WAND_.DLL
        MSVCP100D.DLL
        MSVCR100D.DLL
    FREEGLUT.DLL
    OPENGL32.DLL
    MSVCP110D.DLL
    MSVCR110D.DLL
```

Missing MSVCR100D.DLL

How it appears in Dependency Walker Static Link


Dependency Walker - [StaticLibrary_Test]
File Edit View Options Profile Window Help
STATICLIBRARY_TEST.EXE
MSVCR110D.DLL
KERNEL32.DLL

You don’t see it as its compiled in

Summary

Easier to use static linkage

Sacrifice flexibility and a little space

But, If you change the library need to recompile all apps that use it

Static linking eliminates a host of dynamic problems

- missing dynamic libraries
- libraries that are the wrong version

- Lets build a library

- Set up project (.lib and tester.exe)
- Putting all includes in one place for sharing
- Linking library to tester
- Namespaces – cause you don’t know what other libraries are being used and whether their function/class names are the same as yours
- Then dependencies (build the lib first then the tester)