

49 - Using Libraries in C++ (Static Linking)

sexta-feira, 14 de março de 2025 07:16

- Libraries in c++
- Use external libraries in our code
- Everyone as problems with libs
- Hate the idea of packag manages or link to other repositories
- The idea is to have everything you need to compile and run the projects
- Should be no sync with other repositories
- For c== I just want things to work, clone the repo and compile it
- so, add all the libs we need in out project
- Should I be compiling it my self, or bind in old compiled binary
 - If you don't have access to the source code... link against the binary
 - Otherwise, build it yourself
- Compile our selfs means we can make some changes a well, more freedom

[Using Libraries in C++ \(Static Linking\)](#)



- Using **libraries in C++ with static linking** means that all the code from the library is copied into your executable at compile time. This results in a standalone binary that doesn't depend on the library being present at runtime.
- Static linking **increases executable size** but makes deployment easier since all code is bundled.
- In **Windows**, you might see .lib instead of .a.
 - You don't need the .a or .lib file at runtime—only at compile time.
- Static, the lib is inside the executable when the dinamic is linked in runtime
 - At the application running, oad the ap and you get access to the functions
- Better to use static libs when we can
 - No extra depency to be copied together... like other DLLs with the executable
 - tenicly faster, because of otimiazions because we know what we are linking at that time
 - usually static is the way to go

Using CMake for Static Linking

CMakeLists.txt

```
cmake

cmake_minimum_required(VERSION 3.10)
project(MyApp)

add_library(mathlib STATIC mathlib.cpp)
add_executable(app main.cpp)

target_link_libraries(app PRIVATE mathlib)
```

Then:

```
bash

mkdir build && cd build
cmake ..
make
```

Step 3: Project CMakeLists.txt

```
cmake

cmake_minimum_required(VERSION 3.15)
project(MyGLFWApp)

# Use C++17 or newer
set(CMAKE_CXX_STANDARD 17)

# GLFW static build
add_subdirectory(third_party/glfw)

# Your executable
add_executable(MyApp main.cpp)

# Link GLFW statically
target_link_libraries(MyApp PRIVATE glfw)

# Required on Linux for linking X11 stuff
if (UNIX)
    target_link_libraries(MyApp PRIVATE dl pthread X11 Xrandr Xi Xxf86vm Xcursor)
endif()

# Include GLFW headers
target_include_directories(MyApp PRIVATE third_party/glfw/include)
```

Build GLFW statically

1. Clone GLFW:
git clone <https://github.com/glfw/glfw.git>
cd glfw
2. Build it statically:
cmake -B build -DGLFW_BUILD_DOCS=OFF -DGLFW_BUILD_EXAMPLES=OFF -DGLFW_BUILD_TESTS=OFF -DGLFW_SHARED_LIBS=OFF
cmake --build build
3. Setup Your Project Folder
 - a. place the GLFW binary + headers in a third_party folder:
4. Project CMakeLists.txt
 - a. Follow the exemple n step 3, same structure as used in the exempe
5. Build the Project
mkdir build && cd build
cmake ..
make
6. This will build the dependency locally... and attache to the main project binary
 - a. The include files are under the include "include/GFW/include"
 - b. And the binaries ".a" is under "include/GLFW/build/src"