Name: Mohanad Khaled Hassan Hassan

Track: Professional Development & BI-infused CRM

Information Technology Institute

Dynamic linking and Static linking

When compiling and linking programs in C++, you can choose between two linking methods: static linking and dynamic linking. The main difference lies in when and how libraries are connected to your program.

1. Static Linking

In static linking, all necessary code from libraries is copied directly into the final executable during compilation. The result is a self-contained executable that does not depend on external libraries at runtime.

Pros:

- No dependency on external libraries.
- Slightly faster execution time.

Cons:

- Larger executable file size.
- Must recompile if a library is updated.

2. Dynamic Linking

In dynamic linking, libraries are loaded at runtime from external shared files (e.g., .dll on Windows, .so on Linux, .dylib on macOS).
This creates a smaller executable that depends on the presence of the external shared library.

Pros:

- Smaller executable file.
- Easy to update libraries without recompiling the whole application.
- Shared memory usage between programs.

Cons:

- Requires external libraries to be available at runtime.
- Slightly slower startup time.

3. Example Configuration in C++

Static Linking	Dynamic Linking
Step 1: Compile object file g++ -c mathlib.cpp -o mathlib.o Step 2: Create static library ar rcs libmath.a mathlib.o Step 3: Compile main file with library g++ main.cpp -L. -lmath -o main_static	Step 1: Create shared library g++ -fPIC -shared mathlib.cpp -o libmath.so Step 2: Compile and link main file g++ main.cpp -L. -lmath -o main_dynamic Step 3: Set library path (if needed) export LD_LIBRARY_PATH=.:$LD_LIBRARY_PATH