Static Library

A static library, also known as an archive library, is a collection of precompiled object files bundled together into a single file. It is a type of library that contains compiled code and can be linked with a program at compile time to provide additional functionality.

Static libraries offer several advantages:

1. Code reuse: Static libraries allow you to package commonly used functions, classes, or modules into a single library. This promotes code reuse across multiple projects and avoids duplicating code.

2. Faster linking: Since the object code is already compiled, linking a static library is typically faster compared to linking individual source files. The linker simply needs to extract the necessary object files from the library and incorporate them into the final executable.

3. Standalone executables: When you link a static library, all the code from the library is included directly in the resulting executable. This means that the executable can run independently without requiring the presence of the library on the target system.

4. Portability: Static libraries provide better portability because they eliminate the need for the target system to have the specific library installed. The executable can be distributed as a single file, ensuring that the program will run on any compatible system without worrying about library dependencies.

To create a static library, the following steps are generally involved:

1. Compilation: Compile each source file (.c, .cpp, .f) into object files (.o, .obj) using the appropriate compiler for the language.

2. Archiving: Use a tool like `ar` (Unix-like systems) or `lib` (Windows) to bundle the object files into a static library file. For example, on Unix-like systems, you can create a library called "libmylibrary.a" using the command `ar rcs libmylibrary.a file1.o file2.o`.

3. Linking: Link your main program with the static library during the compilation process. The linker will extract the necessary object files from the library and incorporate them into the final executable. The specific method to link a static library varies depending on the compiler and build system you are using.

When compiling and linking your program, you need to provide the appropriate flags to indicate the location of the library and its name. For example, with GCC, you can use the `-l` flag followed by the library name (e.g., `-lmylibrary`) and `-L` flag followed by the library path (e.g., `-L/path/to/library`).

It's worth noting that when using a static library, the library's code becomes part of the resulting executable. This means that if you make changes to the library, you'll need to recompile and relink your program to incorporate those changes.

Static libraries are useful for small to medium-sized projects, especially when you want to distribute self-contained executables or reuse code across multiple projects without relying on external dependencies.