Object-Oriented Programming

Libraries

Michael L. Collard, Ph.D.

Department of Computer Science, The University of Akron

Libraries

Student writes all the code for an executable program. The only other code used is from standard libraries, which are automatically handled for you.

- On a large program, very rarely will one developer write all the code
- Even if you write the core functionality, your program must adapt to the executable environment and must consider command-line options, multiple input formats, multiple output formats, etc.
- Most programs use *libraries* besides that of the standard libraries
- Most of the libraries used are **not** header-file (include-file) only

- For
acce
cann
- How
part
- Mus

Compile

```
$ g++ -std=c++17 -O3 -c increment.cpp
$ ls -lh *.o
-rw-r--r--  1 collard  admin   520B Apr 28 09:24 increment.o
$ nm --demangle increment.o | grep increment
0000000000000000 T increment(int)
$ g++ -std=c++17 -O3 -c incr.cpp
$ ls -lh *.o
-rw-r--r--  1 collard  admin   4.6K Apr 28 09:25 incr.o
-rw-r--r--  1 collard  admin   520B Apr 28 09:24 increment.o
$ nm --demangle incr.o | grep increment
                 U increment(int)
```

- Compiles a single source file to object code
- Default output filename for object code
- Optimization levels: –01, –02, –03, where higher is faster executable code but takes longer to compile and potentially has a larger object size
- `nm` displays the name list (symbol table) of the object code
- T is the external text (code) section
- Also have to compile *incr.cpp*
- U is externally undefined (does not exist in this file)

```
$ g++ incr.o increment.o -o incr
$ ls -lh incr
-rwxr-xr-x  1 collard  admin    36K Apr 28 09:31 incr
$ nm --demangle incr | grep main
0000000100003978 T _main
$ nm --demangle incr | grep incr
0000000100003dd8 T increment(int)
```

Link

- Creates the executable program `incr`
- All of our source code (in object form) is in the executable
- If we want to use the `increment.cpp` code in another executable, it would be duplicated
- Becomes complicated if there is more than one source-code file that our main program (`incr`) wants to use, e.g., suppose a *decrement.cpp*

Create Static Library

```
$ ar rcs libincrement.a increment.o
$ ls -lh libincrement.a
-rw-r--r--  1 collard  admin   720B Apr 28 09:37 libincrement.a
$ ar -t libincrement.a
__.SYMDEF SORTED
increment.o
```

- A *static library* contains the object code of (potentially) multiple source files
- Existing object code is added to the library
- Static libraries have the extension ".a"
- The name of a static library is prefixed with `lib`
- Use the `ar` command (man ar) to create and view static libraries

Link to Static Library

```
$ g++ incr.o -L. -lincrement -o incr
$ ls -lh incr
-rwxr-xr-x  1 collard  admin    36K Apr 28 10:06 incr
$ nm --demangle incr | grep increment
0000000100003dd8 T increment(int)
```

- The object code for *increment.cpp* is now in the static library *libincrement.a*

- The –L option tells the compiler where the static library files are. In this case, the current directory

- The –l option tells the compiler to link in the object code from the static library, where the prefix *lib* of the library name *libincrement.a* is assumed

Executable with Static Library

```
$ echo "1" | ./incr
2
$ rm libincrement.a
$ echo "1" | ./incr
2
```

- At this point, we can run the program without the static library, as all of our code is in the executable

- If we delete the library or change the *increment.cpp* and recompile, it does not affect our executable `incr`

- Updates to functions from a static library require the executable to be relinked

Static Libraries

- Object files collected into a *library archive* maintained using `ar`
- Uses `.a` extension
- GCC compiler has a standard set of libraries that it includes

  `libc` Standard C library (`glibc` on Linux)

  `libgcc` GCC runtime support library

  `libstdc++` C++ standard library
- Must tell the compiler if you want to use any other libraries with the `-l` option
- Standard library directories: `/lib`, `/usr/lib`, `/lib64`, `/usr/lib64`
- Some Linux distribut
  `/usr/local/lib`.
  distributions do, e.g.
  not, e.g., *Fedora*, *Cent*

Static Libraries Platform Differences

Platform	Typical Compiler	Extension
Linux	gcc	`.a`
macOS	clang	`.a`
Windows	MSVC	`.lib`

Create Shared Library

```
$ g++ -std=c++17 -O3 -fPIC -c increment.cpp
$ ls -lh *.o
-rw-r--r--  1 collard  admin   520B Apr 28 10:10 increment.o
$ g++ -shared increment.o -o libincrement.so
$ ls -lh *.so
-rwxr-xr-x  1 collard  admin    16K Apr 28 10:10 libincrement.so
$ nm --demangle libincrement.so | grep increment
0000000000003fa0 T increment(int)
```

- A *shared library* contains the object code of (potentially) multiple source files
- Difference of a *shared library* with a *static library* is that the object code is **not** linked into the executable, but *shared* at run time
- The option `-shared` makes it a shared library
- Shared libraries on Linux have a *.so* extension (short for *shared object*)
- Shared libraries on macOS have a *.dylib* extension (short for *dynamic library*)

- Obje
cont
Cod

Position Independent Code (PIC)

- Without PIC, the compiled code runs at a fixed memory address
- With PIC, allows functions to be loaded at any position in memory

- Program builds eithe
on, and store in both
or build twice, one w
shared library, and o
the static library.

- Optional for object code in a static library
- Required for object code in a shared library
- Makes accessing functions and global variables indirect
- Calls to code in the library are slightly slower, and the library is slightly larger

Link to Shared Library

```
$ g++ incr.o -L. -lincrement -o incr
$ ls -lh incr
-rwxr-xr-x  1 collard  admin    36K Apr 28 10:06 incr
$ nm --demangle incr | grep increment
                 U increment(int)
```

- The –L option tells the compiler where the shared library files are. In this case, the current directory.

- The –l option tells the compiler to link in the object code from the shared library, where the prefix *lib* of the library name *libincrement.so* is assumed

- The object code is for `increment()` is **not** in the executable `incr`. However, the executable `incr` knows how to call the function

Executable with Shared Library

```
echo "1" | ./incr
./incr: error while loading shared libraries: libincrement.so: cannot open shared object file: No such file or directory
```

- At this point, when we run the program, we get an error

- The problem is that the executable, `incr`, cannot find the shared library and needs the object code for `increment()`

Viewing Shared Library Usage

```
# Linux
ldd incr

# macOS
otool -L incr
```

```
$ldd incr
        linux-vdso.so.1 (0x0000ffff9f180000)
        libincrement.so => not found
        libstdc++.so.6 => /lib/aarch64-linux-gnu/libstdc++.so.6 (0x0000ffff9ef00000)
        libc.so.6 => /lib/aarch64-linux-gnu/libc.so.6 (0x0000ffff9ed50000)
        /lib/ld-linux-aarch64.so.1 (0x0000ffff9f147000)
        libm.so.6 => /lib/aarch64-linux-gnu/libm.so.6 (0x0000ffff9ecb0000)
        libgcc_s.so.1 => /lib/aarch64-linux-gnu/libgcc_s.so.1 (0x0000ffff9ec80000)
```

- To view the use of shared libraries, use the `ldd` command on Linux and the `otool –L` command on macOS
- May have to use `sudo` in front of the `ldd` command
- Note that *libincrement* is "not found"

Temporary Fix for Executable with Shared Library

```
$ echo "1" | LD_LIBRARY_PATH=. ./incr
2
```

- We need to tell the executable where to find the shared library

- The path variable `LD_LIBRARY_PATH` is a **temporary** way of doing this

- Here, we indicate to search in the current directory

Install Shared Library

```
$ cp libincrement.so /usr/local/lib/.
$ echo "1" | ./incr
./incr: error while loading shared libraries: libincrement.so: cannot open shared object file: No such file or directory
$ ldconfig
$ echo "1" | ./incr
2
$ ldd incr
    linux-vdso.so.1 (0x0000ffff8f12f000)
    libincrement.so => /usr/local/lib/libincrement.so (0x0000ffff8f0c0000)
    libstdc++.so.6 => /lib/aarch64-linux-gnu/libstdc++.so.6 (0x0000ffff8ee90000)
    libc.so.6 => /lib/aarch64-linux-gnu/libc.so.6 (0x0000ffff8ece0000)
    /lib/ld-linux-aarch64.so.1 (0x0000ffff8f0f6000)
    libm.so.6 => /lib/aarch64-linux-gnu/libm.so.6 (0x0000ffff8ec40000)
    libgcc_s.so.1 => /lib/aarch64-linux-gnu/libgcc_s.so.1 (0x0000ffff8ec10000)
```

- To install, copy the shared library to a known library directory
- Typically, this is something like `/usr/lib` or `/usr/local/lib`
- May have to use `sudo` for the `cp` command due to the destination location
- Just copying the library is not enough, as the list of libraries is cached by the O.S.
- The command `ldconfig` (man ldconfig) updates this cache
- Note what the `ldd` command now shows

- Insta
the i

Locating Shared Libraries

```
# Standard directories
/usr/local/lib
├── libincrement.so
└── ...

/usr/local/bin
├── incr
└── ...

# Sibling directories
tools/lib
├── libincrement.so
└── ...

tools/bin
├── incr
└── ...

# Same directory
tools
├── libincrement.so
├── incr
└── ...
```

- Standard directories for libraries are the best choice
- Sibling directories often used for alternative package managers, e.g., `brew` uses */opt/homebrew/bin* for Apple Silicon Macs and */opt/homebrew/lib* for Intel Macs (or in *Rosetta2*)
- Same directory often used during development (CMake handles this nicely automatically during builds)
- CMake runtime search path for Linux:

```
set(CMAKE_INSTALL_RPATH "$ORIGIN;$ORIGIN/../lib;/usr/local/lib")
```

- CMake runtime search path for macOS:

```
set(CMAKE_INSTALL_RPATH "@loader_path;@loader_path/../lib;/usr/local/lib")
```

Advantages of Shared Libraries

```
$ ldd incr
    linux-vdso.so.1 (0x0000ffff8f12f000)
    libincrement.so => /usr/local/lib/libincrement.so (0x0000ffff8f0c0000)
    libstdc++.so.6 => /lib/aarch64-linux-gnu/libstdc++.so.6 (0x0000ffff8ee90000)
    libc.so.6 => /lib/aarch64-linux-gnu/libc.so.6 (0x0000ffff8ece0000)
    /lib/ld-linux-aarch64.so.1 (0x0000ffff8f0f6000)
    libm.so.6 => /lib/aarch64-linux-gnu/libm.so.6 (0x0000ffff8ec40000)
    libgcc_s.so.1 => /lib/aarch64-linux-gnu/libgcc_s.so.1 (0x0000ffff8ec10000)
```

- When we update a shared library and rerun the program, we get the new behavior

- All executables share the same library, leading to smaller executable programs

- Easier to update as we install a new shared library, restart the computer, and all executables are using the new library

- Updating a shared library does **not** require the programs to be relinked

Versioning

- User-facing applications/operating systems are versioned with numbers, e.g., *iOS 18*, and often include names, e.g., *macOS Sequoia 15.4.1*
- Sometimes the number is a *build number* or even a *hash*
- Versions of libraries are important as the API may change over time
- Some changes are *breaking changes*, e.g., rename a method
- But not all changes to an API are *breaking changes*, e.g., add new methods
- And we need to update for bug fixes, security enhancements, optimization, etc.

Semantic Versioning

- Form of *MAJOR.MINOR.PATCH*

  MAJOR when you make incompatible API changes

  MINOR when you add functionality in a backward-compatible manner

  PATCH when you make backward-compatible bug fixes

SONAMES

```
$ g++ -shared increment.o -Wl,-soname,libincrement.so.1 -o libincrement.so.1.0.0
$ ls
incr.cpp  incr.o  increment.cpp  increment.hpp  increment.o  libincrement.so.1.0.0
$ cp libincrement.so.1.0.0 /usr/local/lib
$ ls -lh /usr/local/lib/libincrement*
$ cp libincrement.so.1.0.0 /usr/local/lib
$ ls -lh /usr/local/lib/libincrement*
-rwxr-xr-x 1 root root 7.7K Apr 27 12:55 /usr/local/lib/libincrement.so.1.0.0
$ ldconfig
$ ls -lh /usr/local/lib/libincrement*
lrwxrwxrwx 1 root root   21 Apr 27 12:56 /usr/local/lib/libincrement.so.1 -> libincrement.so.1.0.0
-rwxr-xr-x 1 root root 7.7K Apr 27 12:55 /usr/local/lib/libincrement.so.1.0.0
```

- *sonames* are names embedded in a shared library that indicate a particular version
- The name of the shared library is now *libincrement.so.1.0.0*
- The option –`Wl` forwards options to the *linker*
- The linker option –`soname,libincrement.so.1` sets the *soname* field of the shared library to *libincrement.so.1*
- `ldconfig` generates the appropriate symbolic link
- On m
libinc
links
libinc

Packaging

```
$ # Package: libincrement1
$ ls -lh /usr/local/lib/libincrement*
lrwxrwxrwx 1 root root   21 Apr 27 12:56 /usr/local/lib/libincrement.so.1 -> libincrement.so.1.0.0
-rwxr-xr-x 1 root root 7.7K Apr 27 12:55 /usr/local/lib/libincrement.so.1.0.0

# Package: incr
# Dependencies: libincrement1
$ ls -lh /usr/local/bin/incr
-rwxr-xr-x 1 root root 14K Apr 27 12:37 /usr/local/bin/incr

# Package: libincrement1-dev
# Dependencies: libincrement1
$ ls -lh /usr/local/lib/libincrement*
-rw-r--r-- 1 root root 1.4K Apr 27 13:15 /usr/local/lib/libincrement.a
lrwxrwxrwx 1 root root   21 Apr 27 13:15 /usr/local/lib/libincrement.so -> libincrement.so.1.0.0
lrwxrwxrwx 1 root root   21 Apr 27 12:56 /usr/local/lib/libincrement.so.1 -> libincrement.so.1.0.0
-rwxr-xr-x 1 root root 7.7K Apr 27 12:55 /usr/local/lib/libincrement.so.1.0.0
```

- Three packages, *libincrement1* the library, *incr* the application, and *libincrement1-dev* the development package

- The *libincrement1-dev* adds the symbolic link for *libincrement1.so* to get the latest version during development

- The *libincrement1-dev* also adds the static library

- Both packages *incr* and *libincrement1-dev* depend on the package *libincrement1*

Multiple Platforms

```
# Build for increment program

cmake_minimum_required(VERSION 4.2)

project(Increment VERSION 1.0.0)

# incr client
add_executable(incr incr.cpp)
target_compile_features(incr PRIVATE cxx_std_17)
install(TARGETS incr)

# increment static library
add_library(increment_static STATIC increment.cpp)
target_compile_features(increment_static PRIVATE cxx_std_17)
set_target_properties(increment_static PROPERTIES OUTPUT_NAME increment)
install(TARGETS increment_static)

# increment shared library
add_library(increment SHARED increment.cpp)
target_compile_features(increment PRIVATE cxx_std_17)
target_link_libraries(incr PRIVATE increment)
set_target_properties(increment PROPERTIES
    POSITION_INDEPENDENT_CODE ON
    VERSION "${PROJECT_VERSION}"
    SOVERSION "${PROJECT_VERSION_MAJOR}"
)
install(TARGETS increment)

# install include file
include(GNUInstallDirs)
install(FILES ${CMAKE_CURRENT_SOURCE_DIR}/increment.hpp
        DESTINATION ${CMAKE_INSTALL_INCLUDEDIR})
```

- Separate steps for library creation, library installation, and linking executables
- Configurations are different on different platforms, e.g., Linux, macOS, and Windows. This can include filenames, file extensions, directories, etc.

- In W
for a
the s
prog
mult
libra
is to
- All o
betw
e.g.,