Lecture Notes for Lecture 5 of CS 5600 (Computer Systems) for the Fall, 2019 session at the Northeastern University Silicon Valley Campus.

Library Archives

Philip Gust,
Clinical Instructor
Department of Computer Science

Lecture 4 Review

- In this lecture, we continued exploring the capabilities of the bash command shell, including how to apply the software tools concept to usefully combine several tools to create a new one.
- We looked at the command shell as a scripting language, and how to write scripts with variables, control flow and functions utilizing the commands and techniques that we learned about earlier.
- We also learned about the facilities available in the bash shell language for creating full-functioned script applications, including control structures, variables, functions and I/O operations.
- Finally, we saw how to combine script-based and conventional utilities to create more complex applications.

Library Archives

What is a Library?

- A library is a collection of related code and other resources that are packaged together for use by applications.
- The code in a library represents classes and functions that can be combined with the code of an application that calls or instantiates them
- Other resources in a library may also include images, documentation, data, source code, localization files, or any other related resources

Library Archives

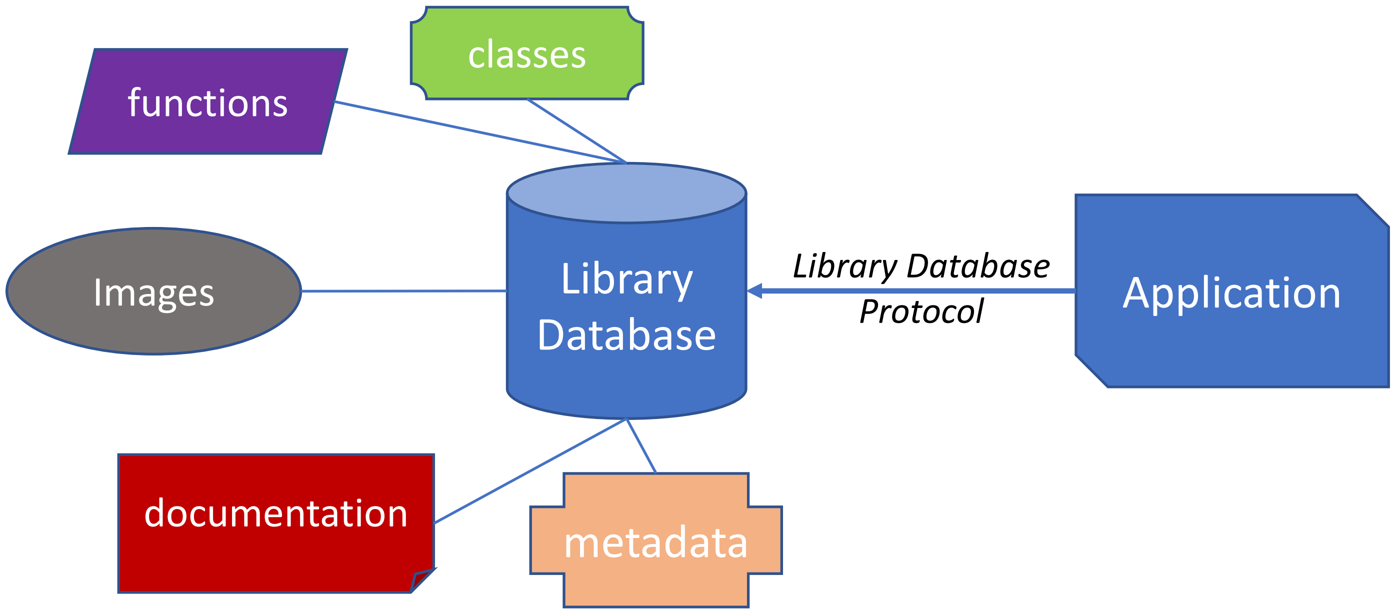
What are their Benefits?

- The value of a library lies in the reuse of the functions and classes and related resources in the library.
- When a program utilizes a library, it gains the behavior implemented inside that library without having to implement that behavior itself.
- Libraries also make it possible for developers to share and reuse code and resources in a modular fashion, and ease their distribution.

Library Archives

How are Library Resources Packaged?

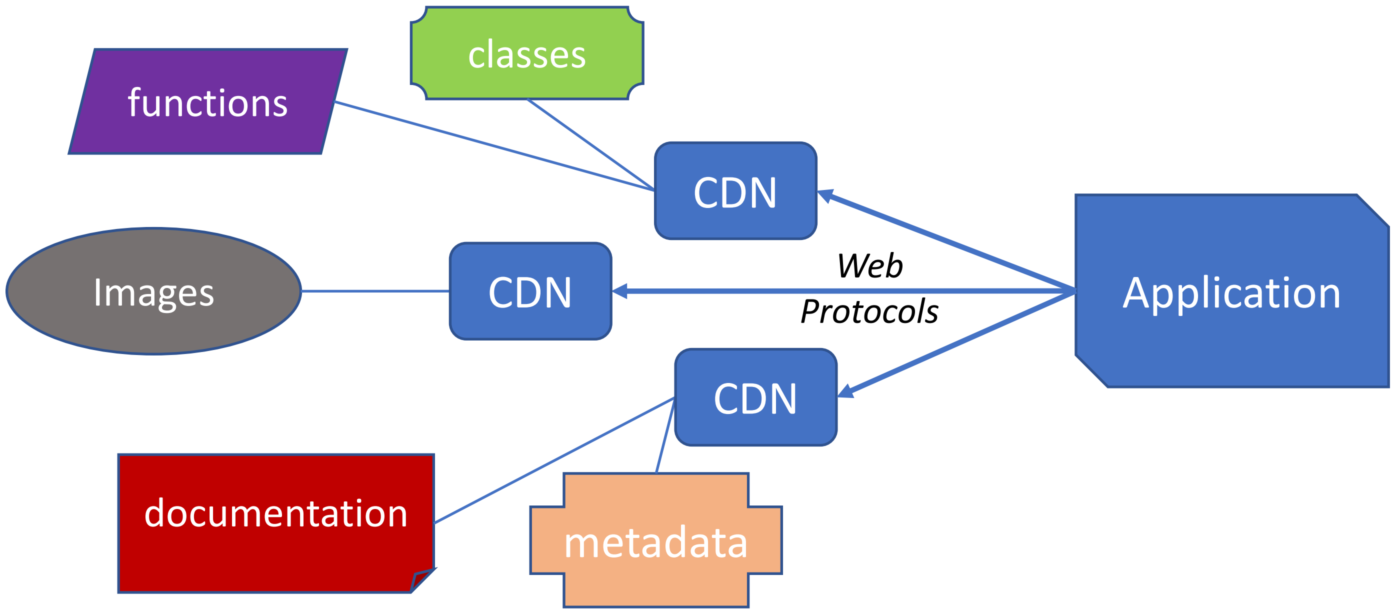
- Library resources can be packaged in one of several ways:
  - In local or remote databases where applications can query and retrieve functions, classes, and other resources. Descriptive metadata facilitates the discovery and use of library content.



Library Archives

How are Library Resources Packaged?

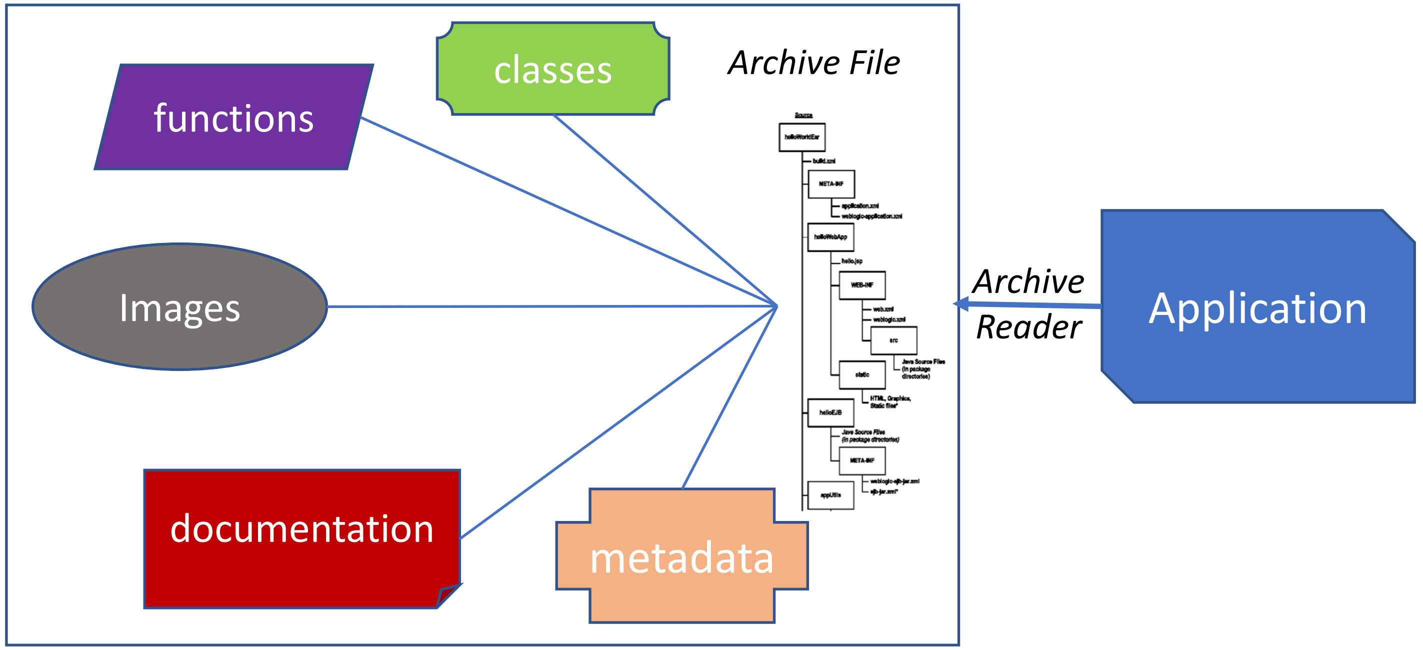
- Library resources can be packaged in one of several ways:
  - In web-based content distribution networks (CDNs) that make remote code and other resources available using web-based discovery and access protocols.



Library Archives

How are Library Resources Packaged?

- Library resources can be packaged in one of several ways:
  - In local archive files that organize the content into directories that parallel the original file system structure, plus manifest files that facilitate finding and using the library content.



Library Archives

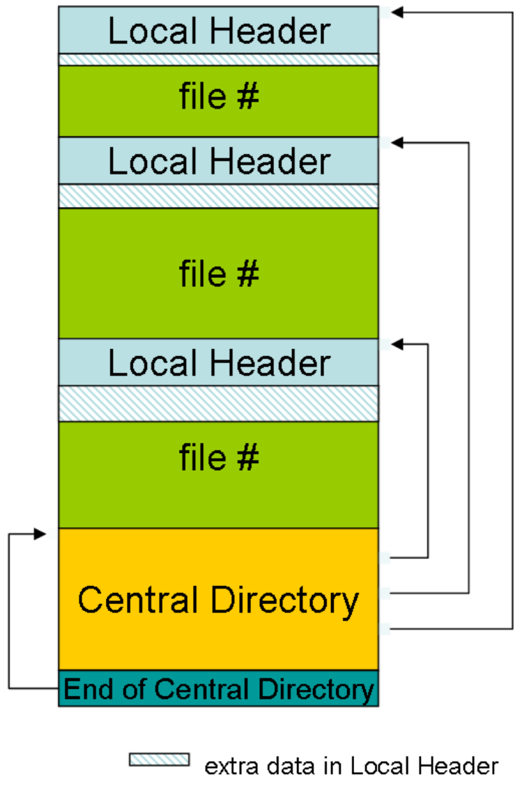
How are Library Resources Packaged?

- In this lecture, we will focus on the library archive file, the most common way to package and distribute code and other resources.
- Most common library archives store all their content in a single file that is formatted internally like a file system.
- The archive format includes regular files and directories where content is stored when the archive is created.
- A special set of functions is available to read content from an archive as though it coming from a regular file.

Library Archives

How are Library Resources Packaged?

- Here is a simple archive file format that illustrates how files can be stored and accessed within the archive file.



Library Archives

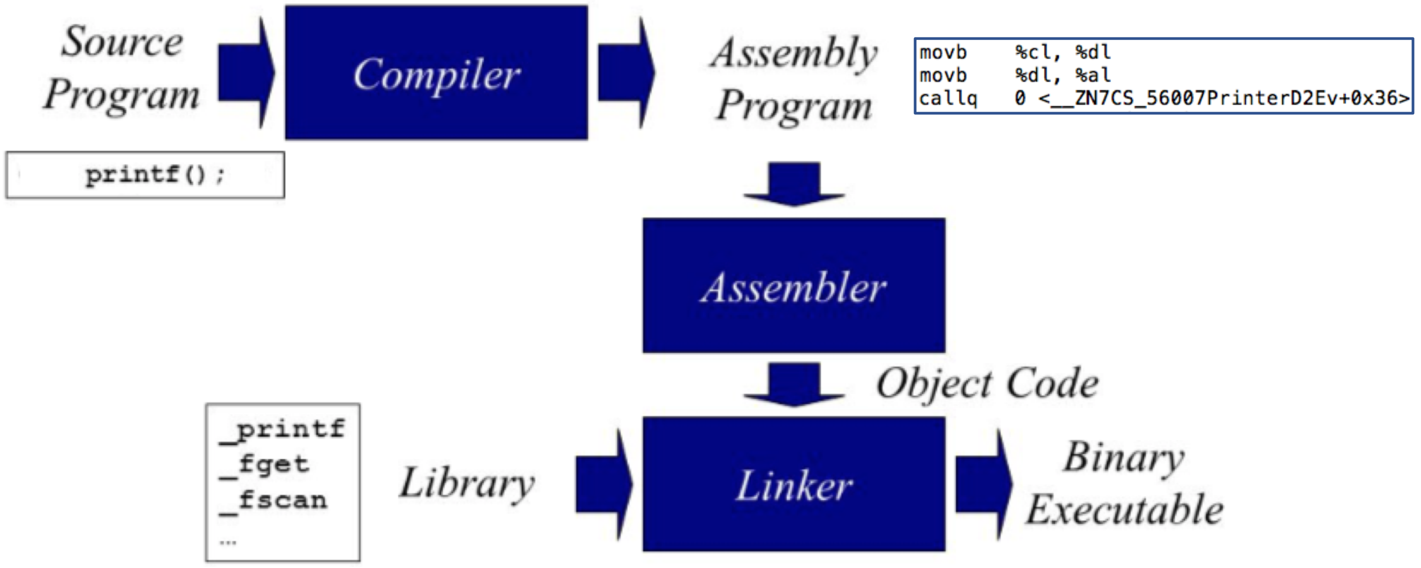
How are Library Archives Created and Used?

- How a library archive is created depends on the programming language. In this lecture, we will look at two cases: C/C++ libraries files, and Java Archive (JAR) files.
- There are some basic similarities in the two kinds of library archives, but there are also some significant difference that are reflected in the languages.

Library Archives

Creating a C/C++ Library Archive

- A C/C++ compiler compiles a source file into an assembly language file that is translated by an assembler into an object file. Object files are linked with library code into an executable.

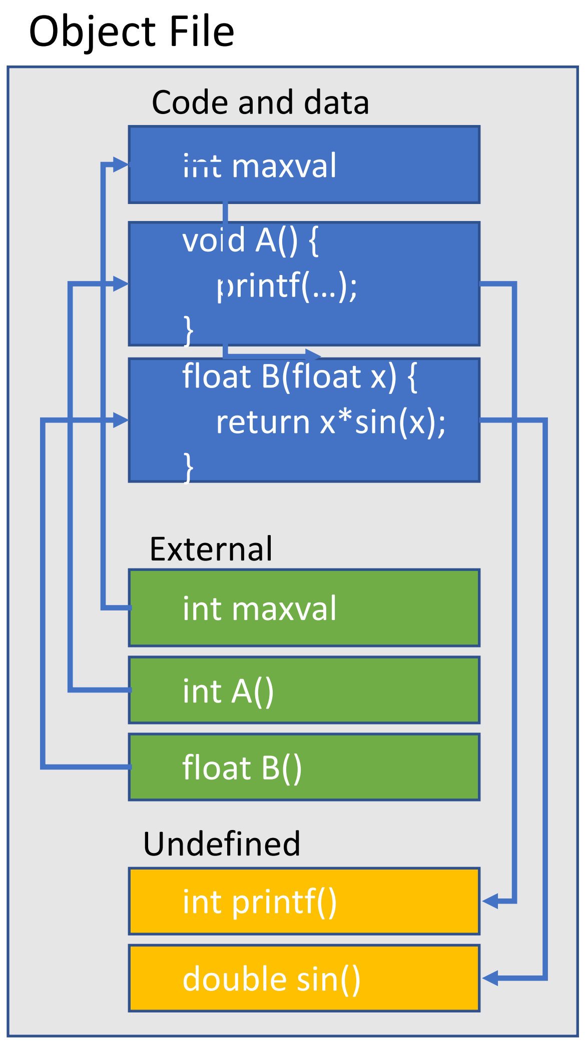


```
movb     %cl, %dl
movb     %dl, %al
callq    0 <__ZN7CS_56007PrinterD2Ev+0x36>
```

Library Archives

Creating a C/C++ Library Archive

- Object files contain compiled code and data, and a *symbol table* with the names of externally visible variables, functions, and classes defined within the code
- The symbol table also includes information about variables, functions, and classes that are referenced by code in the object file but undefined within it.



Library Archives

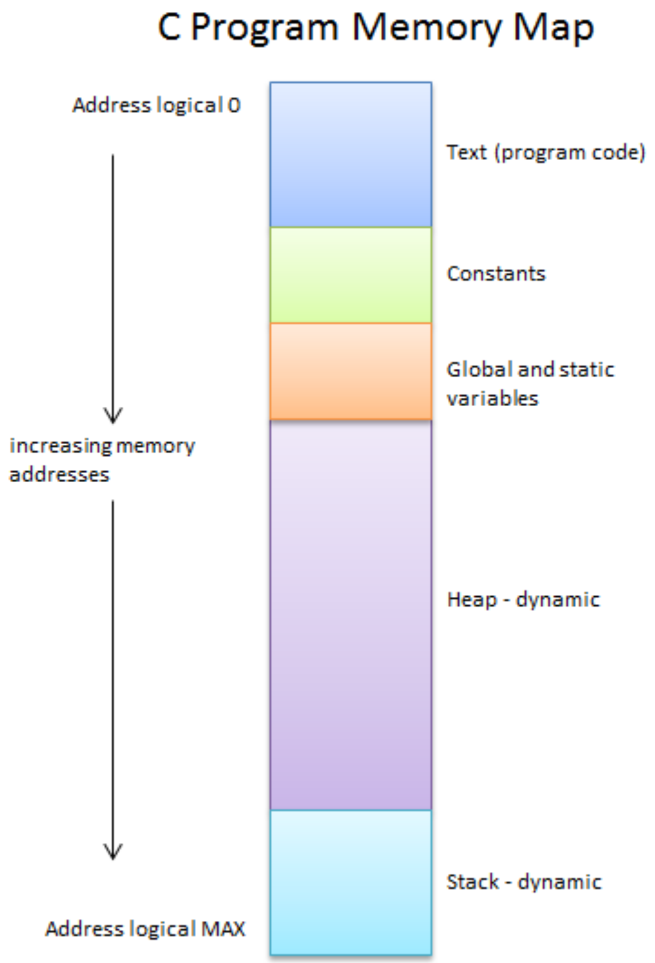
Creating a C/C++ Library Archive

- Most object file formats are structured as separate sections of data, each section containing a certain type of data.
- These sections are known as *segments* due to the term "memory segment", which was a common form of memory management.
- When a program is loaded into memory by a loader, the loader allocates various regions of memory.
- Some of these regions correspond to segments of the object file, and thus are usually known by the same names. Others, such as the stack, only exist at run time.

Library Archives

Creating a C/C++ Library Archive

- Here are the memory segments for a C/C++ program:
  - Text
    - Compiled code for the program
  - Constants
    - Literal strings and other fixed constants
  - Global and static variables
    - Variables declared globally or statically
  - Heap (dynamic memory)
    - A pool of memory programmers can allocate
  - Stack (local variables)
    - Storage for local variables in functions



Library Archives

Creating a C/C++ Library Archive

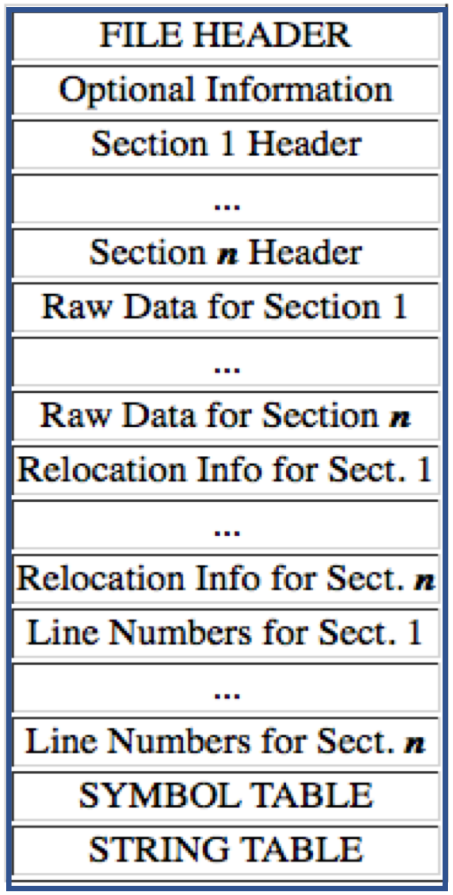
- Typical object file formats reflect these memory segments:
  - Header (descriptive and control information)
  - Code segment ("text segment", executable code)
  - Data segment (initialized static variables)
  - Read-only data segment (rodata, initialized static constants)
  - BSS segment (uninitialized static data, both variables and constants)
  - External definitions and references for linking
  - Relocation information
  - Dynamic linking information
  - Debugging information

Library Archives

Creating a C/C++ Library Archive

- There are several standard object file formats in use in computer systems.
- The Common Object File Format (COFF) was introduced in Unix System V
- COFF and its variants continue to be used on some Unix-like systems, on Microsoft Windows, and in some embedded development systems.
- COFF was largely replaced by the ELF object file format, introduced with SVR4.

COFF Object File

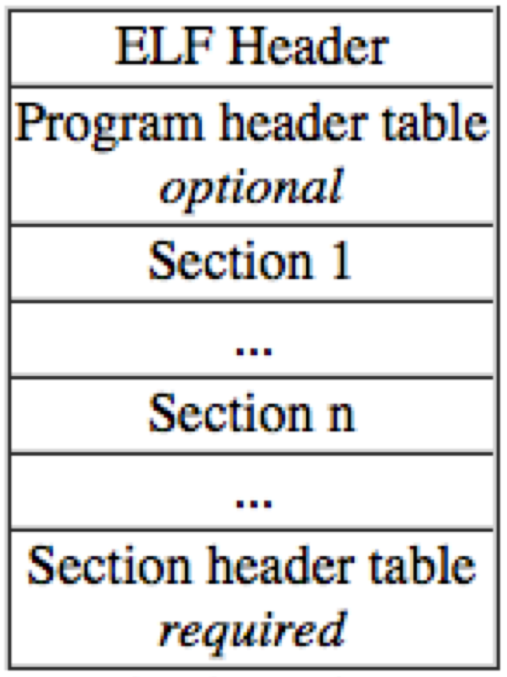


Library Archives

Creating a C/C++ Library Archive

- Executable and Linkable Format (ELF) is a standard file format for executable files, object code, shared libraries. It is extensible and cross-platform.
- It was first published in the specification for Unix System V Release 4 (SVR4).
- ELF is widely accepted among vendors of Unix systems, and was chosen by the 86open project as the standard binary file format for Unix and Unix-like systems on x86 processors, including Linux, and is also used by CygWin under MS Windows.

ELF Object File

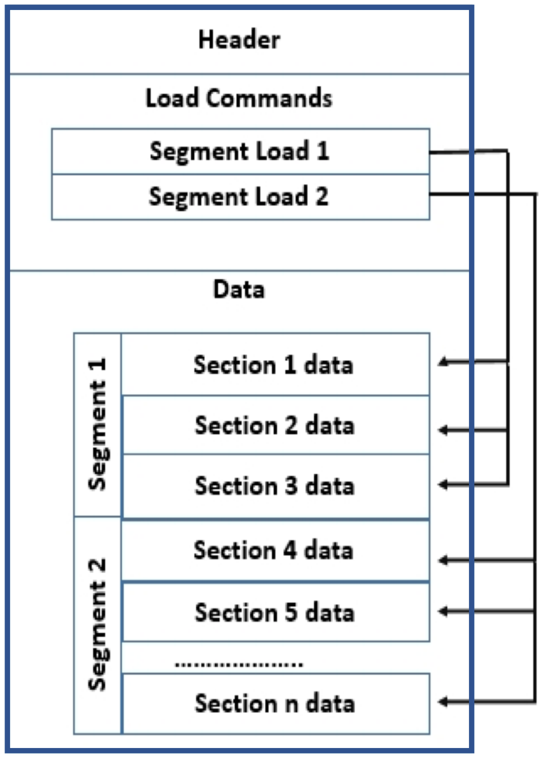


Library Archives

Creating a C/C++ Library Archive

- Mach-O, short for Mach object file format, is a file format for executables, object code, shared libraries, and dynamically-loaded code
- Mach-O offers extensibility and faster access to information in the symbol table.
- Mach-O is used by most systems based on the Mach kernel. MacOS, and iOS are examples of systems that use this format for native executables, libraries and object code.

Mach-O Object File



Library Archives

Creating a C/C++ Library Archive

- The **nm** program can be used to examine the symbols in an object or library archive file.
- We will use nm to examine the object files for a C++ program that implements a printer class and a subclass for a duplex printer variant. The code is in the CCIS GitHub repository 2019FACS5600SV/lecture-5-printer-Cpp
- The **nm** man page describes the options available. Since this is a C++ program, we will use the –C option to demangle the C++ symbol names.

Library Archives

Creating a C/C++ Library Archive

- Here is a description of the symbol types displayed by **nm**.
  - A - Absolute symbol, global
  - a - Absolute symbol, local
  - B - Uninitialized data (bss), global
  - b - Uninitialized data (bss), local
  - D - Initialized data (bbs), global
  - d - Initialized data (bbs), local
  - F - Filename
  - l - Line number entry (see the –a option)
  - S - Section symbol, global
  - s - Section symbol, local
  - T - Text symbol, global
  - t - Text symbol, local (static)
  - U - Undefined symbol

Library Archives

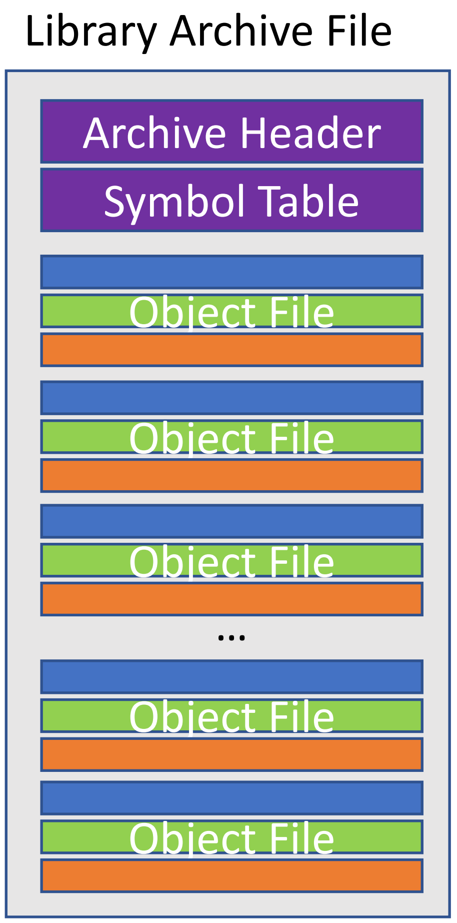
Creating a C/C++ Library Archive

- The **objdump** program can be used to display the assembly language instructions and optionally the original source code in an object file
- The *–disassemble* option causes only the assembly language instruction to be shown.
- The *–source* option causes both the assembly language instructions and the original source code annotations added by the C++ compiler when compiling for debugging.

Library Archives

Creating a C/C++ Library Archive

- A archive program like **ar** combines a group of related object files together into a library archive file.
- For example, the C/C++ runtime functions are distributed in archive files. We have used the archive file for the CUnit unit testing library in our our work.
- Library archives are more convenient to distribute and manage than a large number of individual object files.



Library Archives

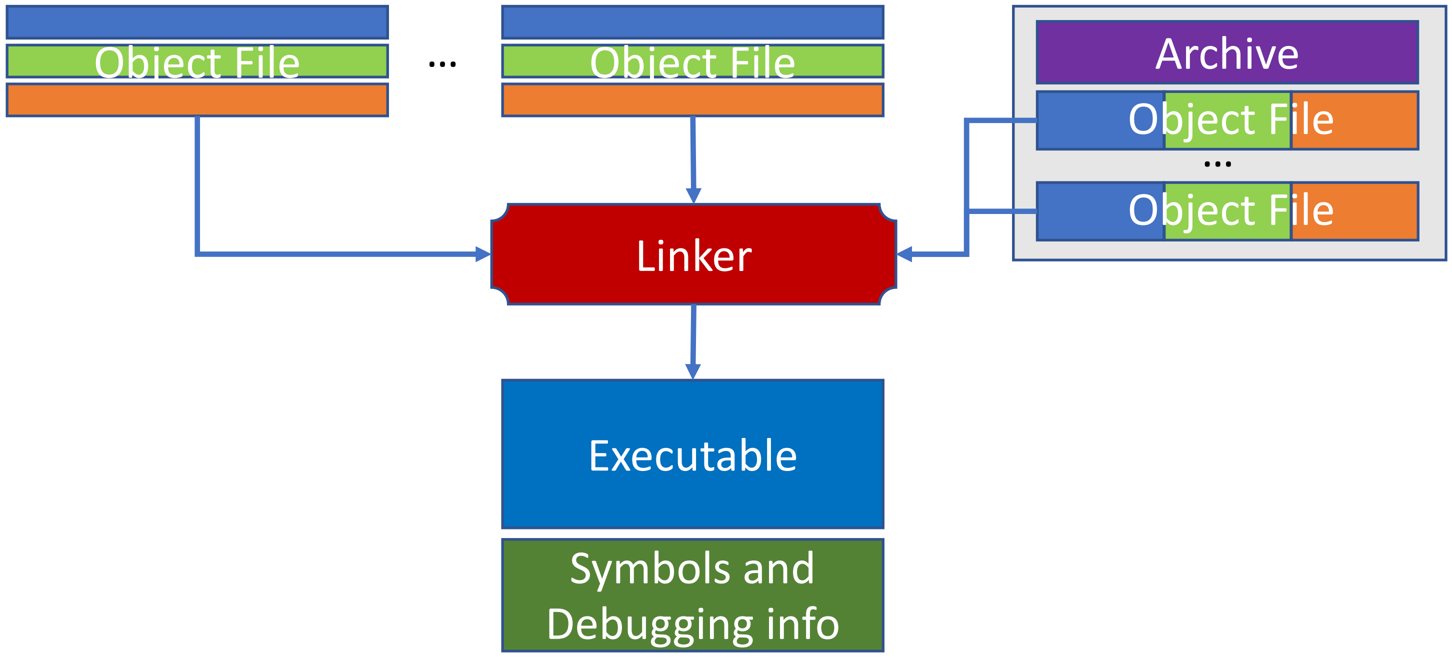
How are C/C++ Library Archives Used?

- A final stage in creating a C/C++ application is linking object files together into an executable program, where all undefined references in each object file are satisfied.
- The linker is given the names of object files to combine into a program, and also the location of archive files that contain object code of commonly used code.
- The linker uses the special functions that can read the object files in library archive files, just as it does regular object files.
- The resulting executable program file contains code and data from the specified object files plus code and data from object files in library archives that satisfy external references.

Library Archives

Creating a C/C++ Library Archive

- Linking C/C++ object files together matches references to external variables, functions and and classes in one object file with definitions contained in other object files.



Library Archives

Static C/C++ Library Archives

- Advantages:
  - All required code and data are available in a single executable program
  - There is no further dependencies on external library archives
  - Statically linked programs are ready to run and require no further processing
- Disadvantages:
  - If a bug is fixed in code within a static library file, all programs linked with the library must be re-linked.
  - Programs take up more memory to run because each one has a copy of the library statically linked.

Library Archives

Static C/C++ Library Archives

- When linking is performed during the creation of an executable, it is known as *static linking* or *early binding*.
- In this case, the linking is usually done by a linker, but may also be done under control of the C/C++ compiler.
- A *static library*, is intended to be statically linked by extracting object files that satisfy external references in the program and adding them in to the executable file.

Library Archives

Shared C/C++ Library Archives

- A *shared library* or *shared object* is a file that is intended to be shared by multiple executables.
- Executables created with shared libraries include a symbol table that enables object files in shared libraries to be incrementally linked to the executable.
- Shared libraries also have additional information that enables them to be incrementally linked to executables.
- Dynamic-Link Library (.dll) format is used on Windows. On MacOS, it is called a Dynamic Library (.dylib). On Linux and other Unix systems it is called a Shared Object (.so).

Library Archives

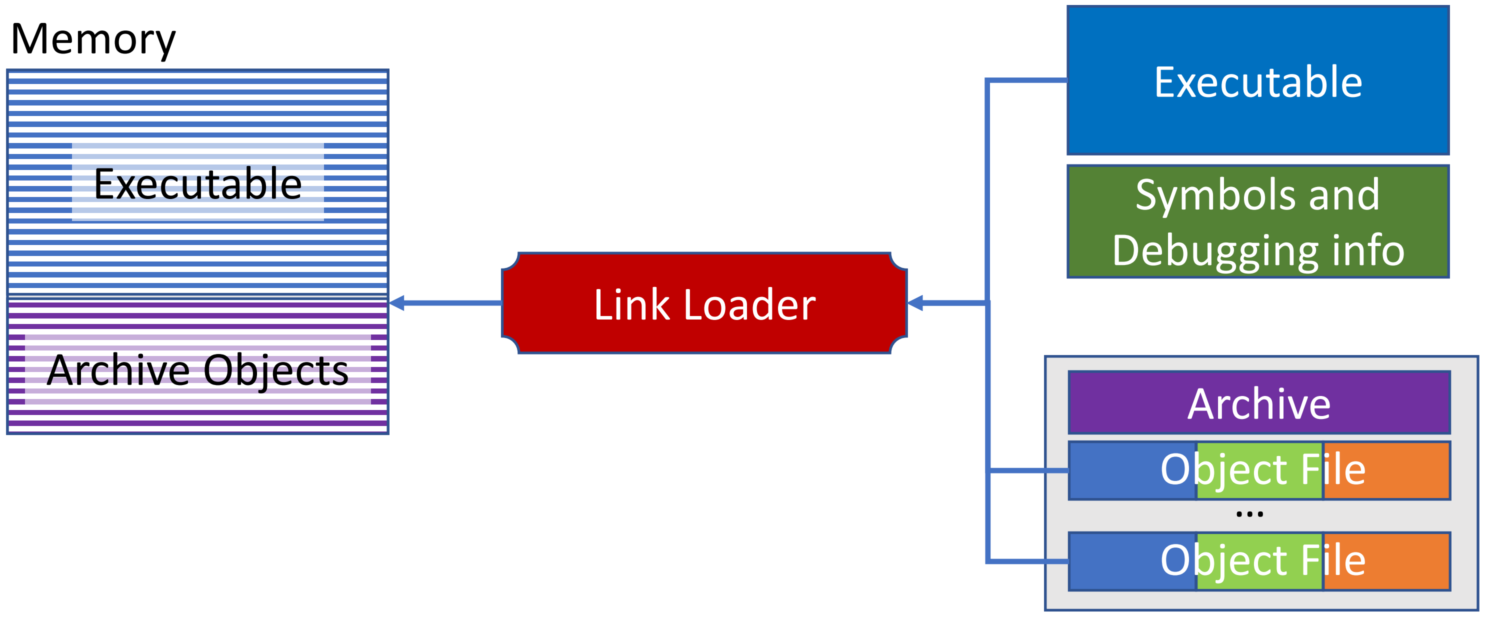
Shared C/C++ Library Archives

- Linking with object files in shared libraries is deferred until an executable is loaded into memory.
- Linking of shared libraries can occur at at one of several times:
  - At load-time, as a program is being loaded into memory
  - At run-time, when a function in an object file is first called by a program
  - At run-time, when a function in an object file is first called by any program running in the system.

Library Archives

Shared C/C++ Library Archives

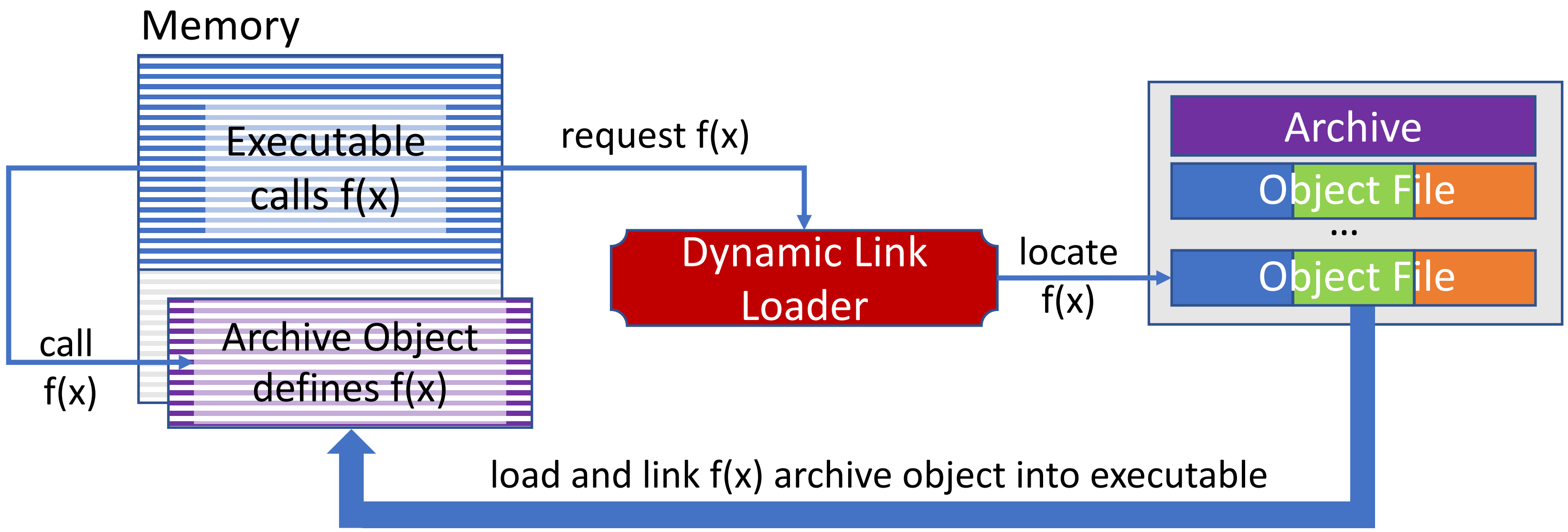
- *Link Loading* occurs as an application is being loaded into memory:



Library Archives

Shared C/C++ Library Archives

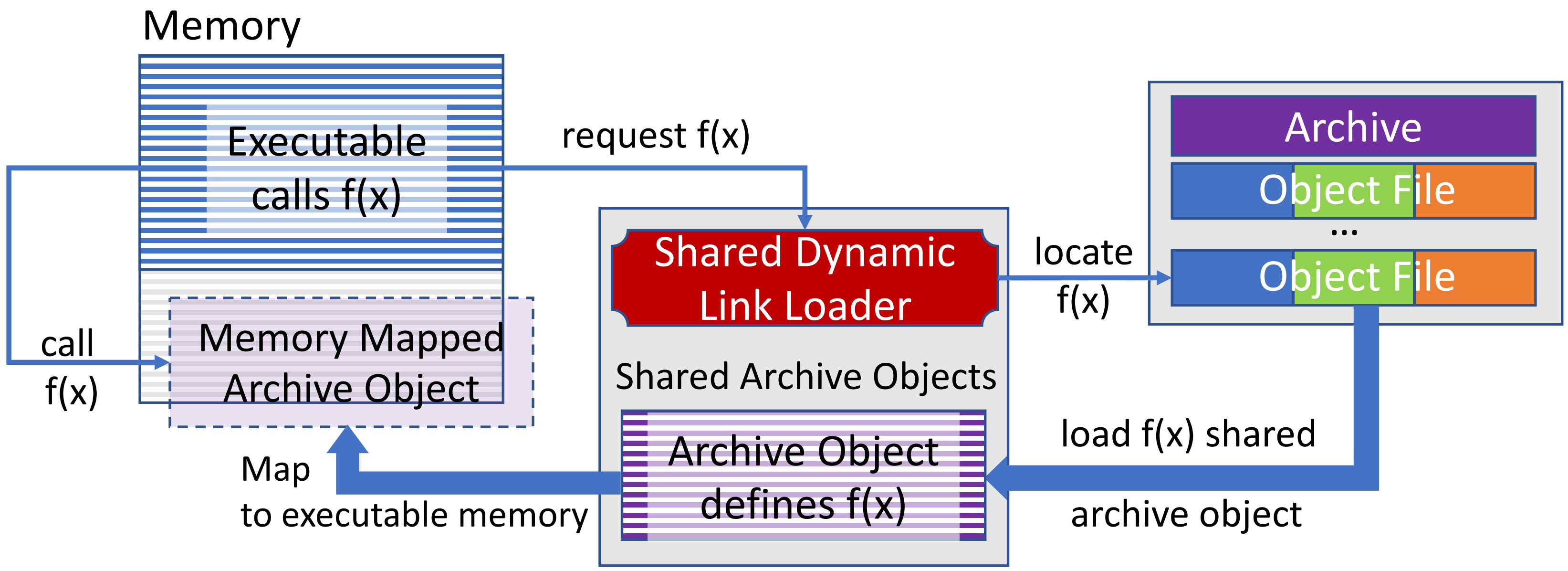
- *Dynamic linking* occurs when an executable calls a function defined in a shared library; object loaded into executable.



Library Archives

Shared C/C++ Library Archives

- *Shared dynamic linking* occurs when an executable calls a function defined in a shared library; mapped to executable.



Library Archives

Shared C/C++ Library Archives

- A dynamic shared archive can be shared in memory by multiple executables. This saves memory space for frequently used archives like the C++ runtime archives.
- On most modern operating systems, shared library files are of the same format as the executable files.
- This offers two main advantages:
  1. It requires making only one loader for both of them, rather than two (having the single loader is considered well worth its added complexity).
  2. It allows the executables also to be used as shared libraries, if they have a symbol table

Library Archives

Creating Static and Shared C/C++ Archive Libraries in Eclipse

- We will walk through the steps for creating both static and shared library versions of the C++ program we saw earlier that implements a printer class and a subclass for a duplex printer.
- The code is in the 2019FACS5600SV/Lecture-5-printer-Cpp CCIS GitHub repository.

Library Archives

Creating Static and Shared C/C++ Archive Libraries in Eclipse

- Create a C++ project “PrintDemo_Cplus”, and a source folder “src” Add the files from the GitHub repo to the “src” folder:
  - DuplexPrinter.cpp
  - DuplexPrinter.h
  - Printer_test.cpp
  - Printer.cpp
  - Printer.h
- Build and verify that the program works correctly. Remember to configure the CUnit include path and library. Now use “Resource Configuration” to exclude “Printer_test.cpp from the build.
- Create another C++ project “TestPrintDemo_Cplus” and a source folder “src”. Copy only ”Printer_test.cpp” to the ”src” folder. Remember to configure the CUnit include path and library.

Library Archives

Creating Static and Shared C/C++ Archive Libraries in Eclipse

- We will create the static library first.
- Go to project *Properties -> C/C++ Build -> Settings -> Build Artifact* tab and change the Artifact Type to "Static Library", and the Artifact Name to ”Printer" (it defaults to the name of the project).
- Clean and rebuild the project and you will see "libPrinter.a" under the project "Archives" subtree.
- Expand the library to verify that it contains “DuplexPrinter.o” and “Printer.o” object files.
- Copy this static library to a new “lib” directory in your home directory and the include files to a new “include” directory. This effectively “distributes” your static library for use in applications.

Library Archives

Creating Static and Shared C/C++ Archive Libraries in Eclipse

- In the “TestPrinterDemo_Cpp” project, add the path to the “include” directory in your home directory to the include paths (e.g. “/Users/phil/include”). Also add the CUnit include path.
- In the “Tool Settings” tab in the *C++ Linker->Miscellaneous* settings, add the full path to the static library under “Other Objects” (e.g. ”/Users/phil/lib/libPrinter.a”).
- Build and run the unit tests and ensure that they pass.
- Congratulations! You have just built and distributed a static library in one project, and linked it with the printer test program in another project.

Library Archives

Creating Static and Shared C/C++ Archive Libraries in Eclipse

- Next we will create the shared library.
- Go to project *Properties -> C/C++ Build -> Settings -> Build Artifact* tab and change the Artifact Type to ”Shared Library", and make sure the Artifact Name is ”Printer". Under “Shared Library Settings”, set the “Shared [-dynamiclib] option.
- Clean and rebuild the project and you will see "libPrinter.dylib" under the project ”Binaries" subtree on MacOS and Linux or "libPrinter.dll" on Windows.
- Expand the library to verify that it contains Entries for all our source files and the files they include.
- Copy this dynamic library to the “lib” directory in your home directory. Your ”include” directory has your include files. This effectively “distributes” your shared library for use in applications.

Library Archives

Creating Static and Shared C/C++ Archive Libraries in Eclipse

- In the “TestPrinterDemo_Cpp” remove the reference to your static library in the *C/C++ Build -> Settings -> Tool Settings* tab in the *C++ Linker -> Miscellaneous* setting.
- Change the *C++ Linker -> Libraries* setting by adding ”Printer" to Libraries and the path to the "lib" directory in your home directory to the Library Search Path.
- Build and run the unit tests and ensure that they pass.
- Congratulations! You have built and distributed a shared library in one project, and linked it with the test program in another project.

  Windows only: Before running the test, you will need to modify in the Run Configuration for your project. In the Environment tab, define the variable "PATH" to the full path to the "lib" directory in your home directory. Windows search the PATH directories for shared libraries.

Library Archives

Creating a Java Archive (JAR)

- A JAR (Java Archive) is a file format used to aggregate many Java class files and associated metadata and resources (text, images, etc.) into one archive file to distribute.
- A JAR file is a file that contains compressed version of .class files, audio files, image files or directories.
- We can imagine a .jar files as a zipped file(.zip) that is created by using ZIP software such as gzip or WinZip.
- Even gzip or WinZip can be used to used to extract the contents of a .jar, So you can use them for lossless data compression, archiving, decompression, and unpacking.

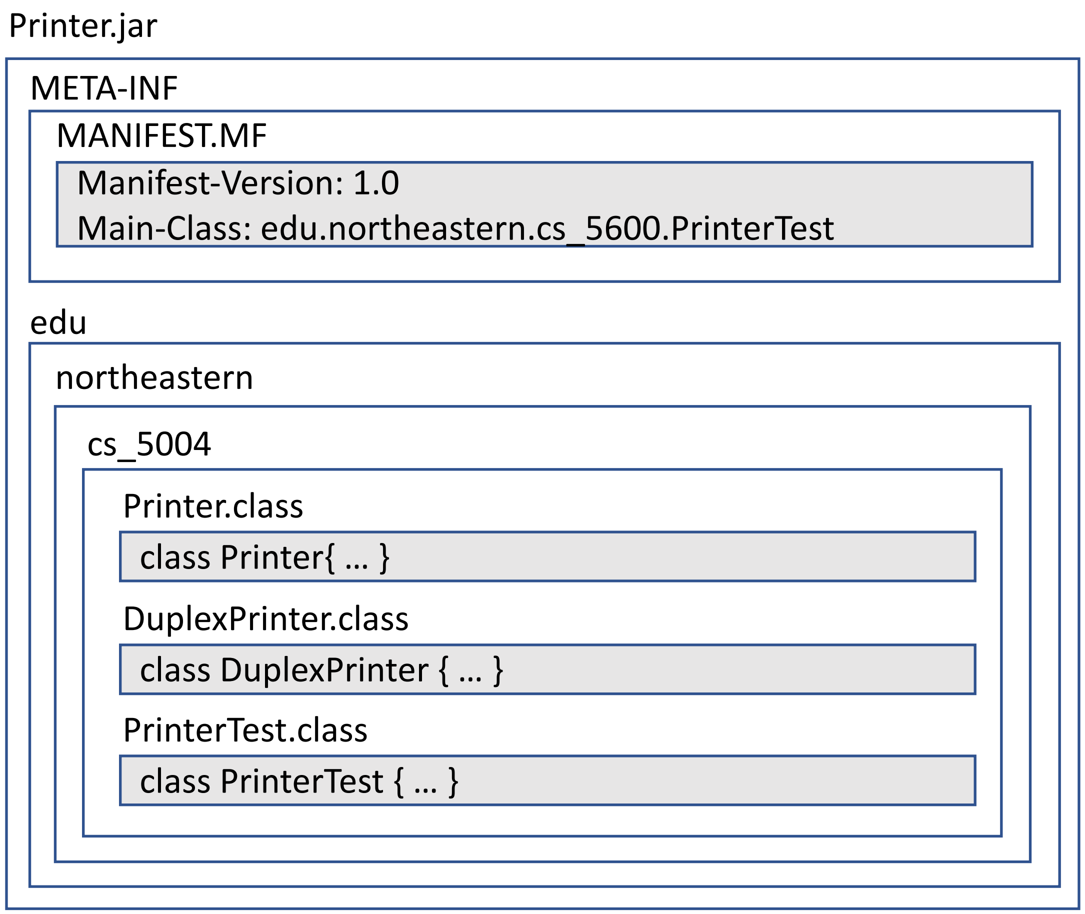
Library Archives

Creating a Java Archive (JAR)

- A Java Archive (JAR) file is a self-contained file systems that includes directories and files.
- Since Java class files are defined in packages, and Java represents the package path as nested subdirectories, the directory structure of a JAR file matches
- The JAR file can also has a manifest file that describes features and metadata of the files in the JAR file.

Library Archives

Creating a Java Archive (JAR)



Library Archives

Creating a Java Archive (JAR)

- The Java runtime environment has classes that enable it to access class files and other resources within a JAR file as easily as it can in a file system
- Java classes and JAR files are always dynamically loaded by the Java runtime as classes are referenced.
- The runtime uses a set of directory paths for regular class files, and JAR file names for JARs. These are maintained in a shell variable named CLASSPATH.
- When a class is required, the Java runtime searches classes in CLASSPATH directories and JAR files in the order specified, and dynamically loads the class.

Library Archives

Creating a Java Archive (JAR)

- The classes required for an application are contained in a set of application specific class files and external JAR files from the Java runtime environment and third-part JAR file
- A JAR file can also be used to package a complete executable Java program that includes classes from external JAR files that the program uses.
- Executable JAR files have the manifest specifying the entry point class with Main-Class: myPrograms.MyClass and an explicit Class-Path (and the -cp argument is ignored).
- Many operating systems can run these directly using the Java runtime environment. They can also be run explicitly using a command like “java -jar foo.jar” from the command line.

Library Archives

Creating Regular and Executable JAR files in Eclipse

- We will walk through the steps for creating both regular and executable versions of the Java version of the printer demo.
- The code is in the 2018FACS5600SV/Lecture-5-printer-Java CCIS GitHub repository.

Library Archives

Creating Regular and Executable JAR files in Eclipse

- Create a Java project “PrinterDemo_Java”. Add the files from the GitHub repo to the “src” folder:
  - DuplexPrinter.java
  - Printer_test.java
  - Printer.java
- Build and verify that the program works correctly. Remember to configure the JUnit JAR file.

Library Archives

Creating Regular and Executable JAR files in Eclipse

- We will create the regular JAR file first.
- Create a JAR file that includes all classes except edu.northeastern.cs_5004.PrinterTest_test by selecting all but that class in your project
- From the pop-up menu, select Export.... Choose *Java->JAR file*, then select "Next".
- Set the export destination to ”Printer.jar" in whatever directory your JUnit JAR files are located in your home directory (e.g. "~/java/lib") and select "Finish” to create the JAR file.

Library Archives

Creating Regular and Executable JAR files in Eclipse

- Create a second project "TestPrinterDemo_Java " with just the edu.northeastern.cs_5600.PrinterTest_test class.
- Under project properties ->Java Build Path, add the two JUnit JAR files and the ”Printer.jar" file you just created as external JARS in the "Libraries" tab.
- Now build and run the unit tests and ensure that it works.
- Congratulations! You have just built and distributed a JAR file in one project, and linked it with the printer test program in another project.

Library Archives

Creating Regular and Executable JAR files in Eclipse

- Now we will create the executable JAR file.
- Create a JAR file by selecting your project, then from the pop-up menu select Export.... Choose *Java->Runnable JAR file*, then select "Next".
- Set the export destination to ”Printer.exec.jar" in whatever directory your JUnit JAR files are located in your home directory (e.g. "~/java/lib").
- Be sure to specify the launch configuration you used to run the program earlier.
- Select "Finish” to create the JAR file.

Library Archives

Creating Regular and Executable JAR files in Eclipse

- *Note*: by default, Eclipse repacks all the classes from JAR files your program requires into the executable JAR file. A dialog reminds you to check that these JAR files allow repacking.
- Some commercial JAR files do not. However, JUnit allows this so you can use this option for this exercise. Check your licenses or contact the vendors.
- Instead, you can choose to package required JAR files into the generated JAR file. Java will also look inside these JAR files to find classes. You should do this if some JAR files that do not allow repacking.

Library Archives

Creating Regular and Executable JAR files in Eclipse

- We can test our executable JAR file by opening a command shell, going to the lib directory where the file was created, and running this command:
  - java -jar Printer.exec.jar
- We should see the same output as from running the program within the Eclipse project.
- Congratulations! You have just built and distributed an executable JAR file that can be run as a standalone executable from Java.