CS4414 Recitation 9

More on Linking and working with open-source libraries

10/24/2025

Alicia Yang

Overview

- C++ compilation and linking
  - Linking review
  - Makefile and Cmake
- Working with open-sourced libraries
  - Llama.cpp

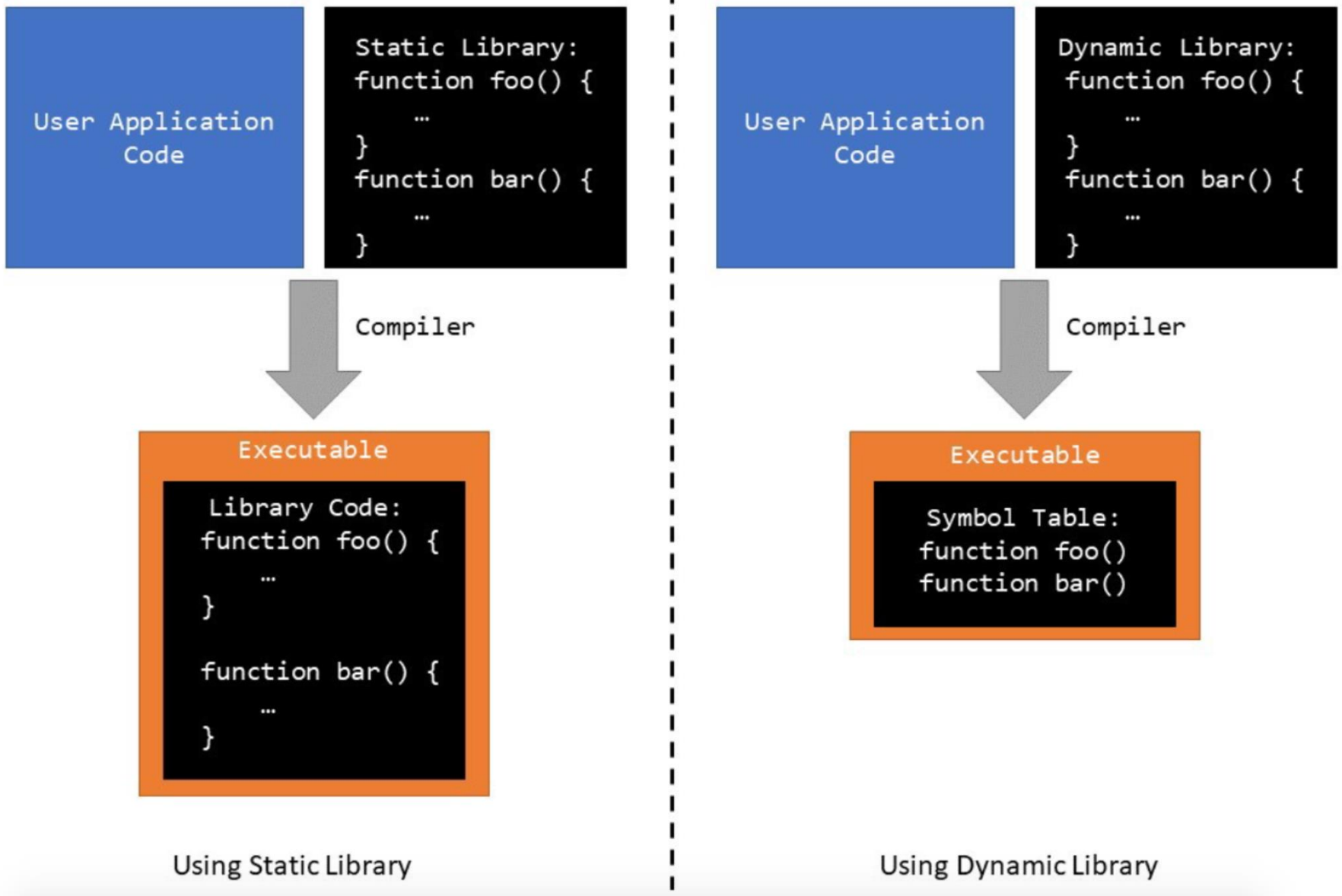
C++ Compilation

- Linking
- Slides from course lecture
  https://www.cs.cornell.edu/courses/cs4414/2024fa/Slides/13-Linking.pdf

Code source:
https://github.com/aliciayuting/CS4414Demo.git

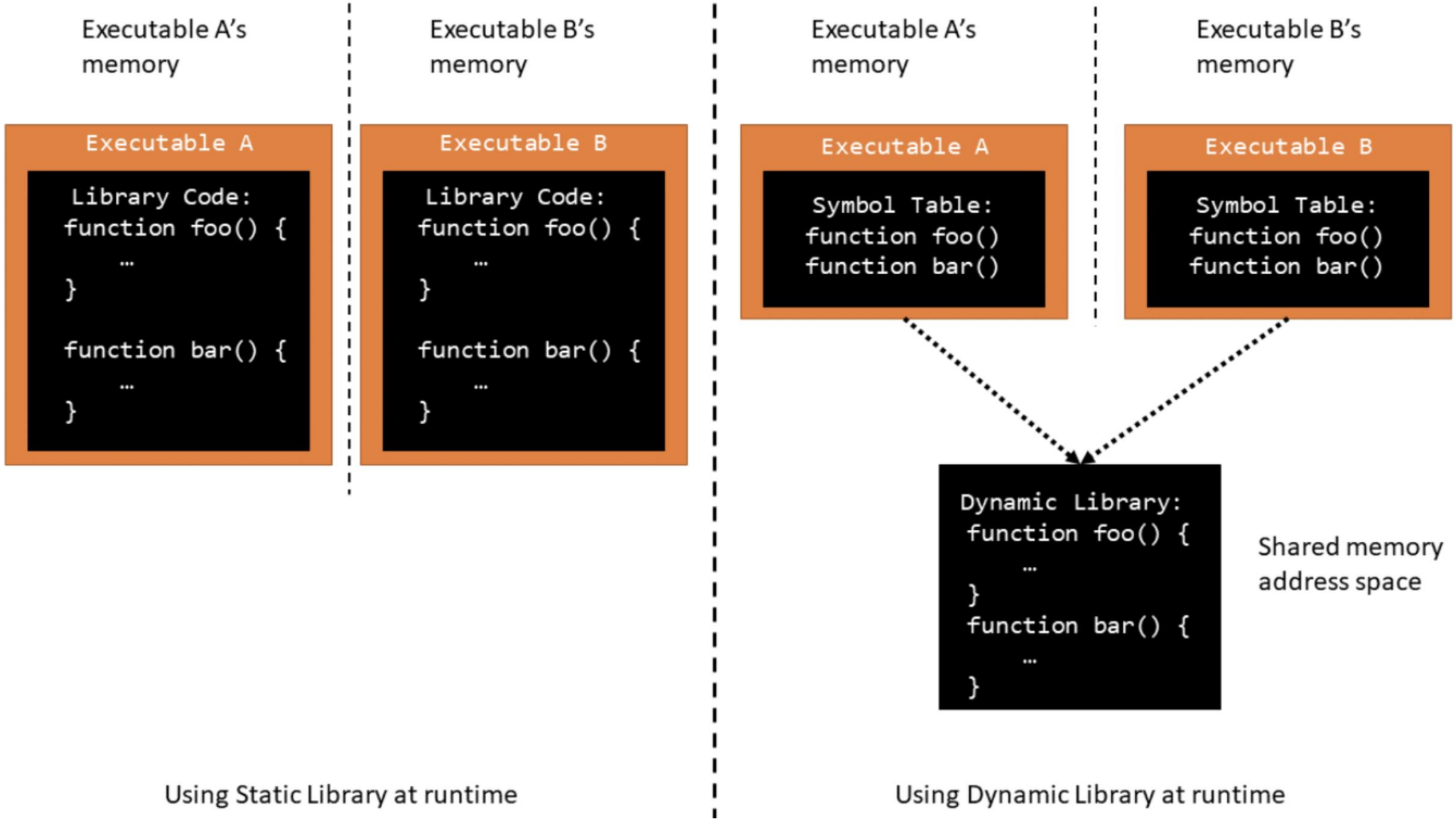
Library Types in C++

--- compile time

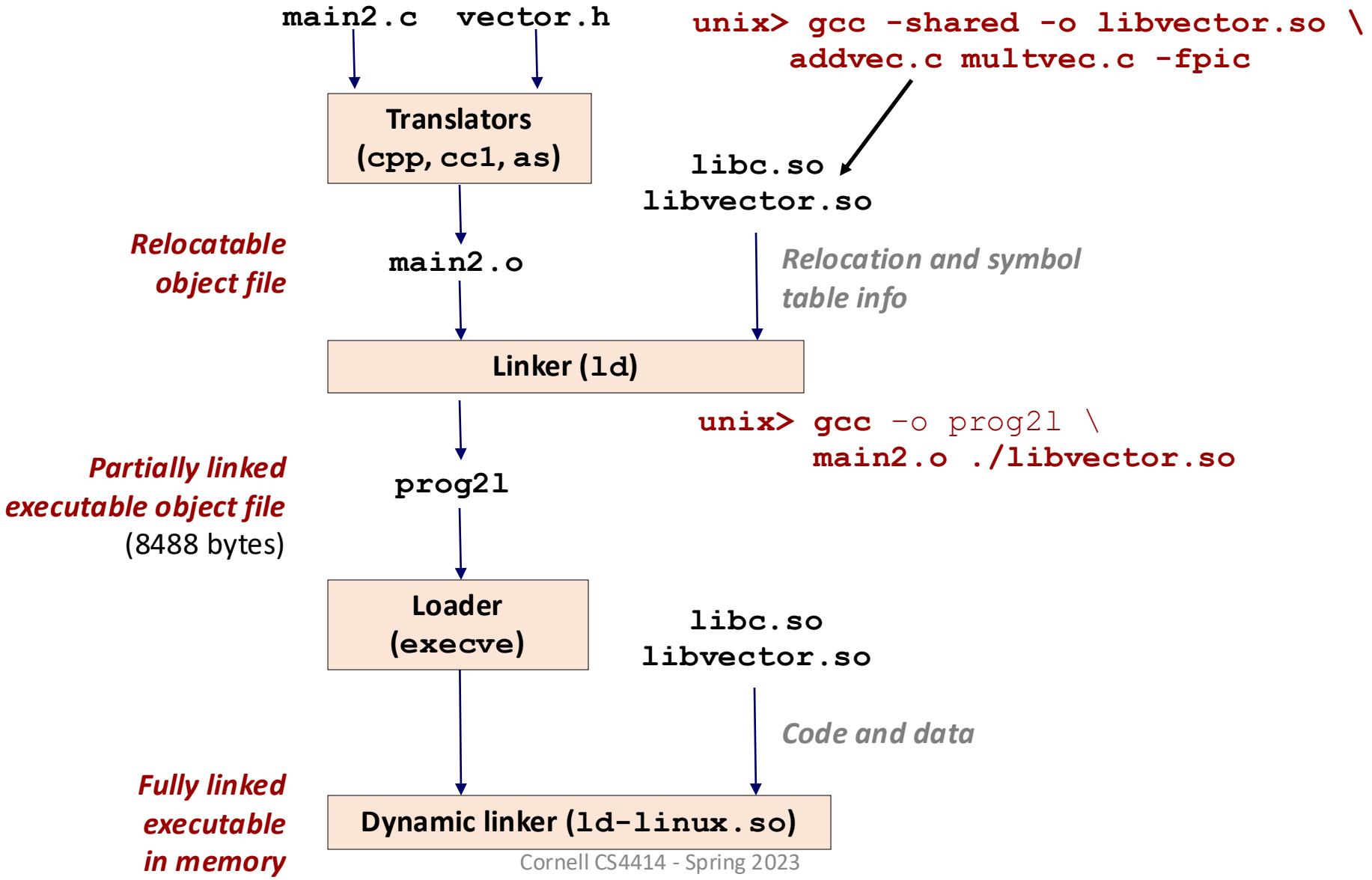


Library Types in C++

--- run time



Dynamic Linking at Load-time



Linking Summary

- Linking is a technique that allows programs to be constructed from multiple object files

- Linking can happen at different times in a program's lifetime:
  - Compile time (when a program is compiled)
  - Load time (when a program is loaded into memory)
  - Run time (while a program is executing)

- Understanding linking can help you avoid nasty errors and make you a better programmer

Getting very fancy: Library Interpositioning (for serious hackers!)

- Documented in Section 7.13 of book
- Library interpositioning: powerful linking technique that allows programmers to intercept calls to arbitrary functions
- Interpositioning can occur at:
  - Compile time: When the source code is compiled
  - Link time: When the relocatable object files are statically linked to form an executable object file
  - Load/run time: When an executable object file is loaded into memory, dynamically linked, and then executed.

1-2-3 Recipe for Interpositioning

- Given an executable that obtains **something** from a library.
- Create a .o file that defines **something**, using the same API the executable expected. Relink the executable against your .o file.
- Now your implementation of **something** will be called

1-2-3 Recipe for Interpositioning

- … but what if you wanted to call the standard **something** from inside your replacement?
- If it were to call **something**, that would just be a recursive call.
- … So, have it call **_something.** This will be undefined… claim that it is in a library

1-2-3 Recipe for Interpositioning

- So now we have the original executable, and it calls your version of **something**, which calls **_something.**

- Create a new DLL library that defines **_something**. It calls the original **something, from the original DLL.**

- Now we have “wrapped” **something**!

… shortcut

- There are also linker arguments you can use to just tell the linker you wish to wrap some method.

- Eliminates the need to create the extra helper DLL.
- Time permitting, I'll show you an example that wraps malloc

Example program

```c
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>

int main(int argc,
         char *argv[])
{
  int i;
  for (i = 1; i < argc; i++) {
    void *p =
          malloc(atoi(argv[i]));
    free(p);
  }
  return(0);
}
```

int.c

- Goal: trace the addresses and sizes of the allocated and freed blocks, without breaking the program, and without modifying the source code.
- Three solutions: interpose on the library `malloc` and `free` functions at compile time, link time, and load/run time.

We won't cover this example if we are short on time; it is not required and you won't see questions about these slides on a quiz

Compile-time Interpositioning

```c
#ifdef COMPILETIME
#include <stdio.h>
#include <malloc.h>

/* malloc wrapper function */
void *mymalloc(size_t size)
{
    void *ptr = malloc(size);
    printf("malloc(%d)=%p\n", (int)size, ptr);
    return ptr;
}

/* free wrapper function */
void myfree(void *ptr)
{
    free(ptr);
    printf("free(%p)\n", ptr);
}
#endif
```

mymalloc.c

Time permitting

Compile-time Interpositioning

```
#define malloc(size) mymalloc(size)
#define free(ptr) myfree(ptr)

void *mymalloc(size_t size);
void myfree(void *ptr);
```

mymalloc.h

```
linux> make intc
gcc -Wall -DCOMPILETIME -c mymalloc.c
gcc -Wall -I. -o intc int.c mymalloc.o
linux> make runc
./intc 10 100 1000
malloc(10)=0x1ba7010
free(0x1ba7010)
malloc(100)=0x1ba7030
free(0x1ba7030)
malloc(1000)=0x1ba70a0
free(0x1ba70a0)
linux>
```

Search for <malloc.h> leads to /usr/include/malloc.h

Search for <malloc.h> leads to

Time permitting

Link-time Interpositioning

```c
#ifdef LINKTIME
#include <stdio.h>

void *__real_malloc(size_t size);
void __real_free(void *ptr);

/* malloc wrapper function */
void *__wrap_malloc(size_t size)
{
    void *ptr = __real_malloc(size); /* Call libc malloc */
    printf("malloc(%d) = %p\n", (int)size, ptr);
    return ptr;
}

/* free wrapper function */
void __wrap_free(void *ptr)
{
    __real_free(ptr); /* Call libc free */
    printf("free(%p)\n", ptr);
}
#endif
```

mymalloc.c

Time permitting

Link-time Interpositioning

```
linux> make intl
gcc -Wall -DLINKTIME -c mymalloc.c
gcc -Wall -c int.c
gcc -Wall -Wl,--wrap,malloc -Wl,--wrap,free -o intl \
    int.o mymalloc.o
linux> make runl
./intl 10 100 1000
malloc(10) = 0x91a010
free(0x91a010)
. . .
```

Search for <malloc.h> leads to /usr/include/malloc.h

- The "-Wl" flag passes argument to linker, replacing each comma with a space.
- The "--wrap,malloc" arg instructs linker to resolve references in a special way:
  - Refs to malloc should be resolved as __wrap_malloc
  - Refs to __real_malloc should be resolved as malloc

Time permitting

Load/Run-time Interpositioning

```c
#ifdef RUNTIME
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <dlfcn.h>

/* malloc wrapper function */
void *malloc(size_t size)
{
    void *(*mallocp)(size_t size);
    char *error;

    mallocp = dlsym(RTLD_NEXT, "malloc"); /* Get addr of libc malloc */
    if ((error = dlerror()) != NULL) {
        fputs(error, stderr);
        exit(1);
    }
    char *ptr = mallocp(size); /* Call libc malloc */
    return ptr;
}
```

Observe that we DON'T have `#include <malloc.h>`

mymalloc.c

Time permitting

Load/Run-time Interpositioning

```
/* free wrapper function */
void free(void *ptr)
{
    void (*freep)(void *) = NULL;
    char *error;

    if (!ptr)
        return;

    freep = dlsym(RTLD_NEXT, "free"); /* Get address of libc free */
    if ((error = dlerror()) != NULL) {
        fputs(error, stderr);
        exit(1);
    }
    freep(ptr); /* Call libc free */}
#endif
```

mymalloc.c

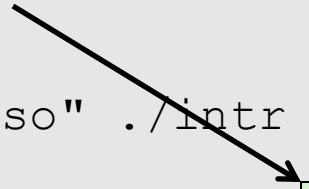
Time permitting

Note: Don’t call printf in malloc/free

- We going overloading malloc…
- … but this means that debugging our code using a printf wouldn’t work: calling anything that does a malloc can cause a recursion that wouldn’t terminate!
- … printf(“something”) turns into std::cout << “something”, and this in turn creates a std::string(“something”). **Program crashes.**

Load/Run-time Interpositioning

```
linux> make intr
gcc -Wall -DRUNTIME -shared -fpic -o mymalloc.so mymalloc.c -ldl
gcc -Wall -o intr int.c
linux> make runr
(LD_PRELOAD="./mymalloc.so" ./intr 10 100 1000)
malloc(10) = 0x91a010
free(0x91a010)
. . .
linux>
```



Search for <malloc.h> leads to /usr/include/malloc.h

- The LD_PRELOAD environment variable tells the dynamic linker to resolve unresolved refs (e.g., to malloc) by looking in mymalloc.so first.
- Type into (some) shells as:

```
env LD_PRELOAD=./mymalloc.so ./intr 10 100 1000)
```

Time permitting

Interpositioning Recap

- Compile Time
  - Apparent calls to **`malloc`**/**`free`** get macro-expanded into calls to **`mymalloc`**/**`myfree`**
  - Simple approach. Must have access to source & recompile
- Link Time
  - Use linker trick to have special name resolutions
    - **`malloc`** → **`__wrap_malloc`**
    - **`__real_malloc`** → **`malloc`**
- Load/Run Time
  - Implement custom version of **`malloc`**/**`free`** that use dynamic linking to load library
    `malloc`/**`free`** under different names
  - Can use with ANY dynamically linked binary

```
env LD_PRELOAD=./mymalloc.so gcc -c int.c)
```

Time permitting

Linking summary

- Usually: Just happens, no big deal
- But there are many sophisticated features and options!
- When using these fancier options, expect strange errors
  - Bad symbol resolution
  - Ordering dependence of linked .o, .a, and .so files
- For power users, it takes effort but then you can do:
  - Interpositioning to trace programs with & without source

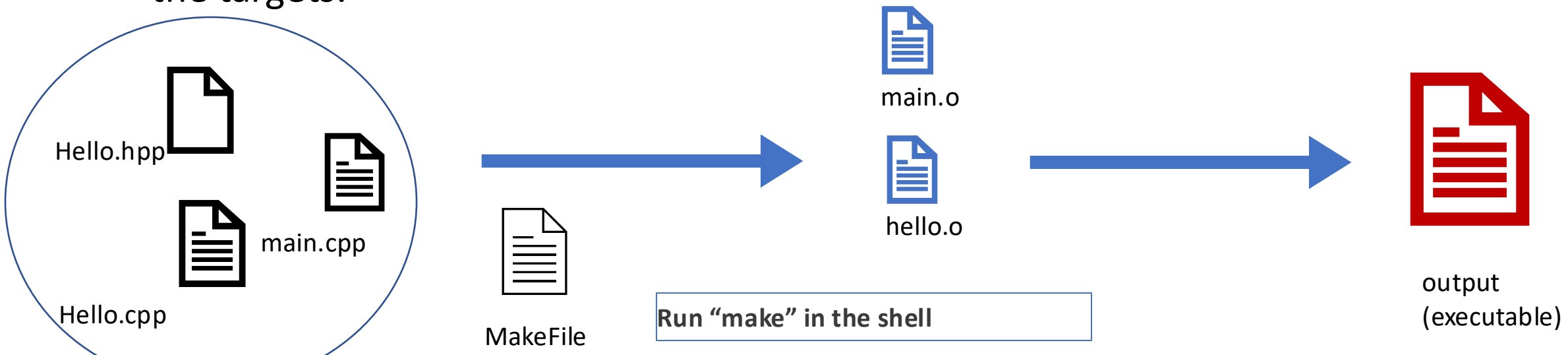
Working with open-sourced library

- CMake recap
- Llama.cpp

Code source:
https://github.com/aliciayuting/CS4414Demo.git

Build Files & Generate Executables --- MakeFile

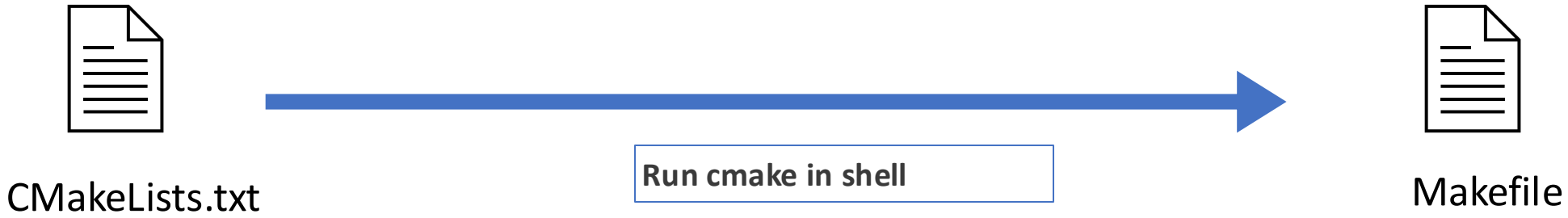
- Makefile is just a text file that is used or referenced by the ‘make’ command to build the targets.




```
CC = g++
CFLAGS = -g -Wall
TARGET = output
all: $(TARGET)
$(TARGET): main.o hello.o
		$(CC) $(CFLAGS) -o $(TARGET) main.o hello.o
main.o: main.cpp hello.hpp
		$(CC) $(CFLAGS) -c main.cpp
hello.o: hello.hpp hello.cpp
		$(CC) $(CFLAGS) -c hello.cpp
```

CMake

CMakeLists.txt files in each source directory are used to generate Makefiles



opencv Public

Sponsor | Watch 2653 | Fork 55.8k | Starred 78.6k

4.x | 6 Branches | 126 Tags | Go to file t | Add file | Code

asmorkalov Merge pull request #26281 from kallaballa:clgl_device_discovery e026a5a · 2 hours ago 34,601 Commits

.github	Force contributors to define Apache 2.0 license for the n...	4 months ago
3rdparty	Merge pull request #26216 from hanliutong:rvv-hal-mer...	last week
apps	Merge pull request #25582 from fengyuentau:dnn/dump...	5 months ago
cmake	Merge pull request #26234 from zachlowry:apply-gcc6-...	last week
data	Merge pull request #22727 from su77ungr:patch-1	2 years ago
doc	Merge pull request #26260 from sturkmen72:upd_doc_...	last week
include	exclude opencv_contrib modules	4 years ago
modules	Merge pull request #26281 from kallaballa:clgl_device_d...	2 hours ago
platforms	Merge pull request #25901 from mshabunin:fix-riscv-aa...	last month
samples	Merge pull request #26212 from jamacias:feature/TickM...	4 days ago
.editorconfig	add .editorconfig	6 years ago
.gitattributes	cmake: generate and install ffmpeg-download.ps1	6 years ago
.gitignore	Merge pull request #17165 from komakai:objc-binding	4 years ago
CMakeLists.txt	build: set cmake policy for if(IN_LIST) support	2 days ago
CONTRIBUTING.md	migration: github.com/opencv/opencv	8 years ago

About

Open Source Computer Vision Library

opencv.org

opencv | c-plus-plus | computer-vision | deep-learning | image-processing

Readme
Apache-2.0 license
Security policy
Activity
Custom properties
78.6k stars
2.7k watching
55.8k forks
Report repository

Releases 63

OpenCV 4.10.0 Latest
on Jun 3

+ 62 releases

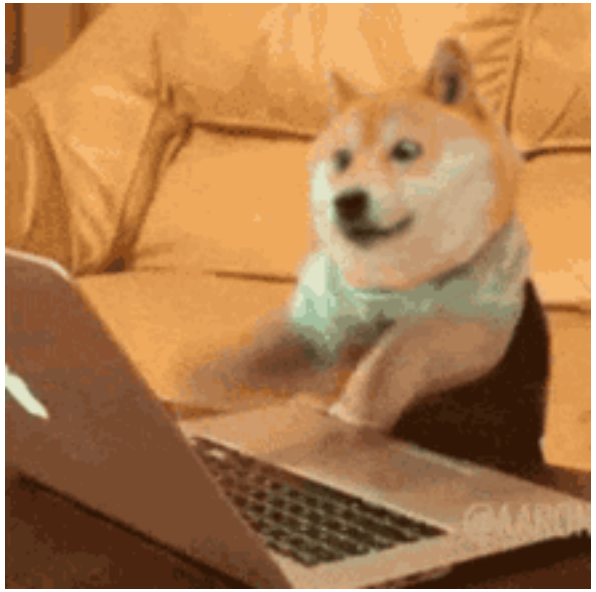
Sponsor this project

opencv OpenCV

Sponsor

Demo with Llama.cpp

https://github.com/ggml-org/llama.cpp.git



Steps

https://github.com/aliciayuting/CS4414Demo/blob/main/recitation9/README.md

1. Build and install llama.cpp library
2. Choose a model and download (For memory consideration, we recommand TinyLlama-1.1B-Chat-v0.3-GGUF)
3. Use llama-cli to test the model
4. Write a cpp program via llama.cpp API
5. Compile the program and run

Explaining the compilation

```
g++ -std=c++20 run_llama.cpp \
  -I"$HOME/opt_dev/include" \
  -L"$HOME/opt_dev/lib" \
  -Wl,-rpath,"$HOME/opt_dev/lib" \
  -lllama -lggml -lggml-base -lggml-cpu \
  -lpthread -ldl -lm \
  -o run
```

Define the include path.
When code includes
#include llama.h

Compile time, add library search path.
Tells the linker (during build) where to find the libraries I want to link against.
e.g. when I use –lama, the linker expands it to $HOME/opt_dev/lib/libllama.so (shared)

Runtime loader
Tells the dynamic loader (during execution) where to find the .so
$HOME/opt_dev/lib/libllama.so (shared)

Explaining the compilation

```
g++ -std=c++20 run_llama.cpp \
  -I"$HOME/opt_dev/include" \
  -L"$HOME/opt_dev/lib" \
  -Wl,-rpath,"$HOME/opt_dev/lib" \
  -lllama -lggml -lggml-base -lggml-cpu \
  -lpthread -ldl -lm \
  -o run
```

Links the program to libllama.so

Link with libggml.so
GGML is a llama.cpp's tensor / math engine.

-ldl
link against the dynamic loader, used for:
- Runtime dynamic library loading
- Symbol lookups

-lm
link with math library

Where to find the resources?

- Course webpage CS4414 https://www.cs.cornell.edu/courses/cs4414/2025fa/
- Recitation demo:

  https://github.com/aliciayuting/CS4414Demo/tree/main/recitation9