Compiling Code Bases

Generating an Executable File

C File (.c)

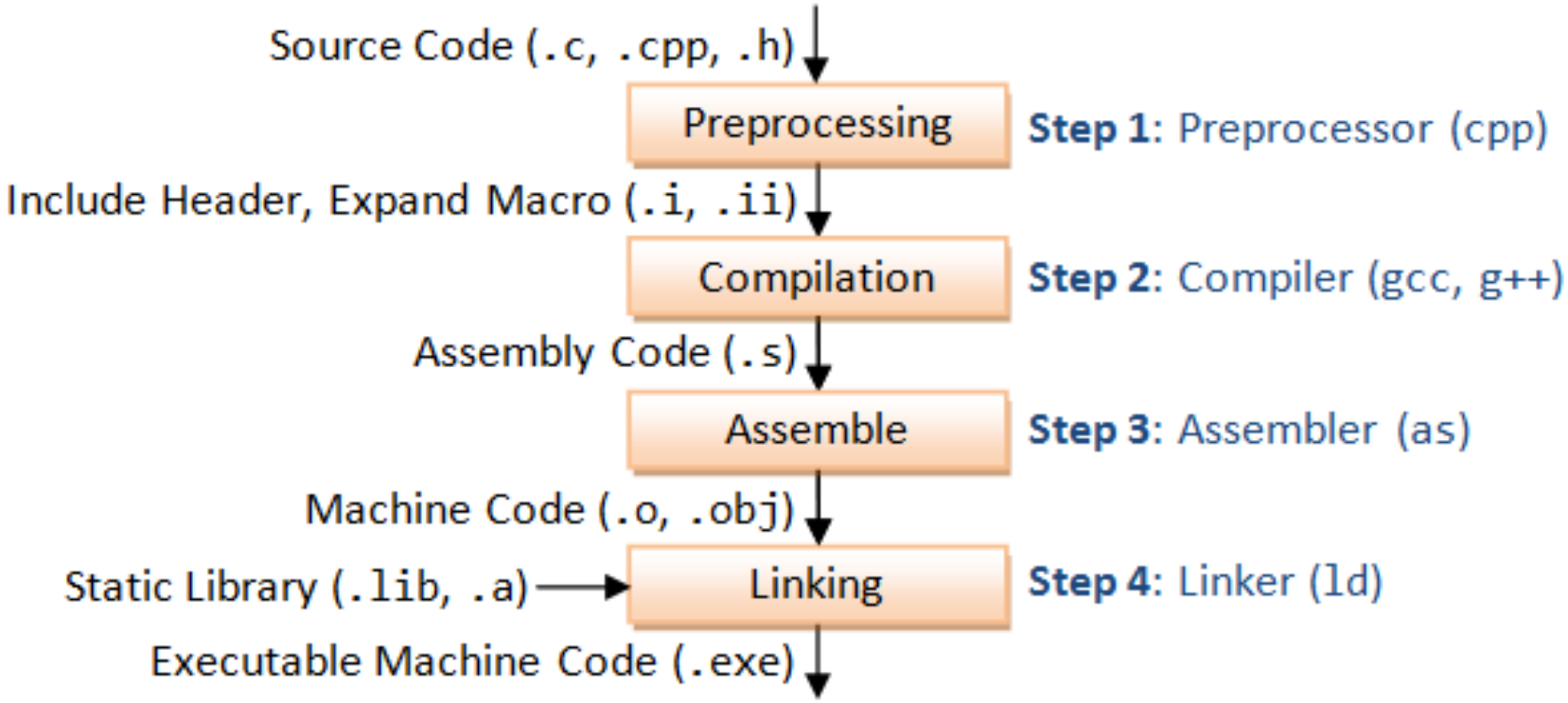
Compiler: translate from human readable to machine-specific code

Object File (.o)

- Intermediate machine-specific representation of just what is in a C file

Linker: bring together multiple object files so that all functions are known

Executable (no extension)


Source Code (.c, .cpp, .h)
Preprocessing
Step 1: Preprocessor (cpp)
Include Header, Expand Macro (.i, .ii)
Compilation
Step 2: Compiler (gcc, g++)
Assembly Code (.s)
Assemble
Step 3: Assembler (as)
Machine Code (.o, .obj)
Static Library (.lib, .a)
Linking
Step 4: Linker (ld)
Executable Machine Code (.exe)

Gnu C Compiler (gcc)

- Performs the compiling and linking phases for us
- Also invokes the assembler as part of the compiling process

Compiling Code Bases

As the set of files in a program gets large, we want to:

- Have a way to invoke the compiler easily
- Only compile the code that needs to be compiled
- Have a way to track which files depend on which other files

Invoking gcc at the compiler gets tiring and error prone…

Make Files

One of several ways to manage the compiling/project management process

- Define dependencies: what files depend on other files?
- Define rules for how to create derived files
  - Including the invocation of the compiler
- Uses file time stamps to know what work actually needs to be done

Our First Program

```
#include <stdio.h>

int main(int argc, char** argv)
{
  printf("Hello, World\n");
}
```

gcc hello.c –o hello

Our First Makefile

```
# The top rule is executed by default
all: hello

# Other rules are invoked as necessary

# Rule for creating the hello executable
hello: hello.c
    gcc hello.c -o hello
```

Automatic Variables

Automatic variables are set by make after a rule is matched. There include:

- $@: the target filename.
- $*: the target filename without the file extension.
- $<: the first prerequisite filename.
- $^: the filenames of all the prerequisites, separated by spaces, discard duplicates.
- $+: similar to $^, but includes duplicates.
- $?: the names of all prerequisites that are newer than the target, separated by spaces.

For example, we can rewrite the earlier makefile as:

```
all: hello.exe

# $@ matches the target; $< matches the first dependent
hello.exe: hello.o
	gcc -o $@ $<

hello.o: hello.c
	gcc -c $<

clean:
	rm hello.o hello.exe
```