### **Illustrating Makefile Commands with a Step-by-Step Example**

We will create a **Makefile** for the following project structure:

/home/labDirectory/

├── file1.cpp (Contains `main()`)

├── file2.cpp (Contains `main()`)

├── lib1.cpp (No `main()`, compiled into object file)

├── lib2.cpp (No `main()`, compiled into object file)

├── lib3.cpp (No `main()`, compiled into object file)

├── lib1.h

├── lib2.h

├── lib3.h

├── Makefile

## **1️⃣ Understanding the Requirements**

* We **compile** the 3 non-main() C++ files into object (.o) files.
* We **link** these object files with file1.cpp and file2.cpp to create **two executables**.
* The compilation uses:
  + --std=c++11 **when running make normally.**
  + --std=c++11 -Wall -Werror **if additional flags are given (make CXXFLAGS="...").**
* The final directory **must contain only**:
  + **Original 9 source files**
  + **3 .o files**
  + **2 executables**

## **2️⃣ Writing the Makefile**

# Compiler settings

CXX := g++

CXXFLAGS := --std=c++11 # Default flag (can be overridden)

# Source files

SRCS := file1.cpp file2.cpp lib1.cpp lib2.cpp lib3.cpp

HEADERS := lib1.h lib2.h lib3.h

# Object files (only for non-main files)

OBJS := lib1.o lib2.o lib3.o

# Executables

EXE1 := file1.out

EXE2 := file2.out

# Default target (Runs when `make` is executed)

all: $(EXE1) $(EXE2)

# Rule for creating executables (each depends on object files)

$(EXE1): file1.cpp $(OBJS)

$(CXX) $(CXXFLAGS) -o $@ $^

$(EXE2): file2.cpp $(OBJS)

$(CXX) $(CXXFLAGS) -o $@ $^

# Rule for compiling object files

%.o: %.cpp $(HEADERS)

$(CXX) $(CXXFLAGS) -c $< -o $@

# Clean target (removes generated files)

clean:

rm -f $(OBJS) $(EXE1) $(EXE2)

## **3️⃣ Explanation of Each Rule**

### **🔹 all: $(EXE1) $(EXE2)**

* **Default target**. When you run make, it compiles both file1.out and file2.out.

### **🔹 $(EXE1): file1.cpp $(OBJS)**

* **Creates file1.out**
* Uses file1.cpp and the three object files.
* $@ → The target (i.e., file1.out).
* $^ → All dependencies (file1.cpp lib1.o lib2.o lib3.o).

### **🔹 $(EXE2): file2.cpp $(OBJS)**

* Same as above but for file2.out.

### **🔹 %.o: %.cpp $(HEADERS)**

* This rule **compiles each .cpp file into a .o file**.
* % is a **pattern-matching wildcard**:
  + If lib1.cpp needs to be compiled, this rule will create lib1.o.

### **🔹 clean Target**

Deletes object files and executables to reset the build:  
 make clean

## **4️⃣ Running the Makefile**

### **(a) Default Compilation**

make

* Uses CXXFLAGS="--std=c++11"

### **(b) Custom Compilation Flags**

make CXXFLAGS="--std=c++11 -Wall -Werror"

* Uses **new flags** without modifying the Makefile.

### **(c) Cleaning the Directory**

make clean

* Removes all .o files and executables.

## **5️⃣ Breakdown of Makefile Features**

| **Feature** | **Explanation** | **Example** |
| --- | --- | --- |
| CXX := g++ | Defines the compiler | g++ |
| CXXFLAGS := --std=c++11 | Default compilation flags | --std=c++11 |
| all: | Default target | make |
| $@ | Target name | file1.out |
| $^ | All dependencies | file1.cpp lib1.o lib2.o lib3.o |
| %.o: %.cpp | Pattern rule for object files | lib1.cpp → lib1.o |
| clean: | Removes generated files | make clean |

## **6️⃣ Final Directory Structure After Running Make**

/home/labDirectory/

├── file1.cpp

├── file2.cpp

├── lib1.cpp

├── lib2.cpp

├── lib3.cpp

├── lib1.h

├── lib2.h

├── lib3.h

├── Makefile

├── file1.out (Executable)

├── file2.out (Executable)

├── lib1.o (Object file)

├── lib2.o (Object file)

├── lib3.o (Object file)

## **Conclusion**

🔹 This **Makefile correctly handles dependencies** between files.  
 🔹 It ensures only the required files exist after compilation.  
 🔹 Using **pattern rules (%.o: %.cpp)**, it efficiently **compiles object files**.  
 🔹 The **flexible CXXFLAGS setting** allows different compilation flags dynamically.

Would you like an even more advanced feature like **dependency tracking**? 🚀