Introduction

This guide provides a brief introduction to creating, using, and modifying Makefiles to compile and link C++ projects.

1 Understanding the Project Structure

In this example, we have a C++ project with the following files:

- `main.cpp` — Contains the `main()` function that drives the program.
- `Class.hpp` — The header file for the `Class` class, which declares its members and methods.
- `Class.cpp` — The implementation file for the `Class` class, defining the methods declared in `Class.hpp`.

Note: The files `Class.hpp` and `Class.cpp` are paired, with `Class.hpp` containing the interface (class declaration) and `Class.cpp` containing the implementation.

2 Starting the Makefile

We'll start by specifying the compiler and compiler flags:

1. Open a text editor and create a new file named `Makefile`.
2. Define the C++ compiler and the flags used during compilation:

   ```
   CXX = g++
   CXXFLAGS = -std=c++17 -g -Wall -O2
   ```

   Explanation: `CXX` specifies the C++ compiler (`g++`), and `CXXFLAGS` defines the flags for the compiler:

   - `-std=c++17` — Specifies that the C++17 standard should be used.
   - `-g` — Includes debugging information in the compiled files.
   - `-Wall` — Enables all compiler warnings.
   - `-O2` — Enables a level of optimization to improve performance.

3 Defining the Target and Object Files

Next, we define the target program and the object files:

1. Specify the name of the final executable and the object files:

   ```
   PROG ?= main
   OBJS = Class.o main.o
   ```

   Explanation: `PROG` is the name of the final executable, and `OBJS` lists the object files that need to be linked to create the executable. Here, `Class.o` corresponds to the compiled `Class.cpp` file, and `main.o` corresponds to the compiled `main.cpp` file.

4 Compiling Source Files

We now define the rule for compiling the source files into object files:

1. Define the rule for converting `.cpp` files to `.o` files:

   ```
   .cpp.o:
   $(CXX) $(CXXFLAGS) -c -o $@ $<
   ```

   Explanation: This rule tells `make` how to compile `.cpp` files into object files (`.o` files). `$<` is the source file (e.g., `Class.cpp`), and `$@` is the output file (e.g., `Class.o`).

5 Linking Object Files

Next, we define the rule for linking the object files into the final executable:

1. Define the rule for building the executable:

   ```
   $(PROG): $(OBJS)
   $(CXX) $(CXXFLAGS) -o $@ $(OBJS)
   ```

   Explanation: This rule tells `make` to create the executable by linking the object files (`OBJS`). `$(PROG)` is the target executable, and `$(OBJS)` are the object files to be linked.

6 Adding Clean and Rebuild Rules

Finally, we add rules for cleaning up the build directory and rebuilding the project:

1. Add a `clean` rule to remove generated files:

   ```
   clean:
   rm -rf *.o main
   ```

   Explanation: The `clean` rule deletes all object files and the executable.

2. Add a `rebuild` rule to clean and then rebuild the project:

   ```
   rebuild:  clean all
   ```

   Explanation: The `rebuild` rule first runs the `clean` rule and then rebuilds the project by running the `all` target.

7 Final Makefile

Here is the complete Makefile for our C++ project:

```
CXX = g++
CXXFLAGS = -std=c++17 -g -Wall -O2

PROG ?= main
OBJS = Class.o main.o

all: $(PROG)

.cpp.o:
 $(CXX) $(CXXFLAGS) -c -o $@ $<

$(PROG): $(OBJS)
 $(CXX) $(CXXFLAGS) -o $@ $(OBJS)

clean:
 rm -rf *.o main

rebuild: clean all
```

Explanation: This Makefile automates the entire process of compiling and linking a C++ project. The commands are general enough to be reused for different projects by simply changing the file names and target name.

8 Using the Makefile

1. To build the project, navigate to the directory containing the Makefile and type:

   ```
   make
   ```

2. To clean the project, removing object files and the executable, type:

   ```
   make clean
   ```

3. To rebuild the project from scratch, type:

   ```
   make rebuild
   ```

9 Extending the Makefile

In this section, we'll modify the Makefile to handle a subclass that inherits from the base class `Class` and change the output executable to `program`.

Assume the following additional files are present in the project:

- `Subclass.hpp` — The header file for the subclass `Subclass`, which inherits from `Class`.
- `Subclass.cpp` — The implementation file for `Subclass`.

We'll update the Makefile to compile and link these additional files, resulting in an executable named `program`.

Updated Makefile:

```
CXX = g++
CXXFLAGS = -std=c++17 -g -Wall -O2

PROG ?= program
OBJS = Class.o Subclass.o main.o

all: $(PROG)

.cpp.o:
 $(CXX) $(CXXFLAGS) -c -o $@ $<

$(PROG): $(OBJS)
 $(CXX) $(CXXFLAGS) -o $@ $(OBJS)

clean:
 rm -rf *.o program

rebuild: clean all
```

Explanation:

- `PROG` is updated to `program`, so the final executable will be named `program`.
- `OBJS` now includes `Subclass.o`, which is the object file for `Subclass.cpp`.
- The `clean` rule is updated to remove the new executable `program`.

Using the Updated Makefile:

- Build the project by running `make` as before. The executable will now be named `program`.
- Clean the project by running `make clean`, which will remove all object files and the `program` executable.

10 References

GNU Make Documentation: https://www.gnu.org/software/make/manual/make.html
Makefile Tutorial: https://makefiletutorial.com/
C++ Programming Resources: https://cplusplus.com/