

C++ For C Coders 8

Build Process

Data Structures
C++ for C Coders

한동대학교 김영섭교수
idebtor@gmail.com

build process
compile & link
static library
make & makefile

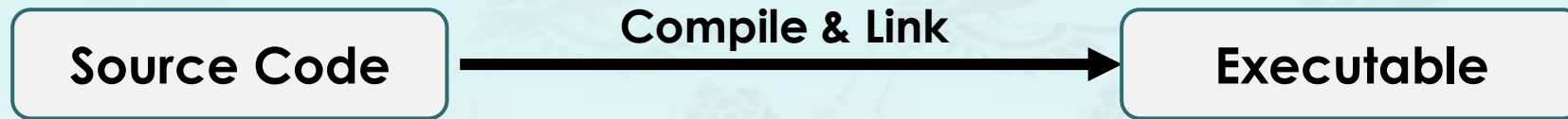
Build Process

1. Compile & Link
2. Build Process
3. Make and makefile
4. Build a static library

Build Process

The term build process here refers to the steps starting with source code (**a set of .cpp and .h files**) ending with an executable file representing your program.

- A simplistic view of the build process

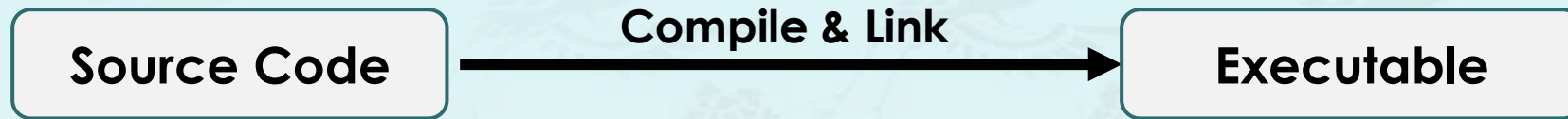


```
> g++ quick.cpp comparator.cpp printlist.cpp -I../include -o qsort
```

Build Process

The term build process here refers to the steps starting with source code (**a set of .cpp and .h files**) ending with an executable file representing your program.

- A simplistic view of the build process



> g++ quick.cpp comparator.cpp printlist.cpp -I../include -o qsort

- A simple but realistic view of the build process



> g++ -c quick.cpp comparator.cpp printlist.cpp -I../include

> g++ quick.o comparator.o printlist.o -L../lib -lsort -o qsort

Compile

Link

The two command lines can be combined into one line shown below:

> g++ quick.cpp comparator.cpp printlist.cpp -I../include -L../lib -lsort -o qsort
(Assume that **../lib/libsort.a** exists.)

Compile & Link

Building an executable for a program consists of two major stages:

- **Compile stage (.cpp, .h → .o)**
 - **Syntax** checked for correctness.
 - Variables and function calls checked to ensure that correct declarations were made and that they match.
 - It translates the source code into **object code**. It is not an executable.
 - It **does not match function definitions** to their calls at this point.

Compile & Link

Building an executable for a program consists of two major stages:

- **Compile stage (.cpp, .h → .o)**
 - **Syntax** checked for correctness.
 - Variables and function calls checked to ensure that correct declarations were made and that they match.
 - It translates the source code into **object code**. It is not an executable.
 - It **does not match function definitions** to their calls at this point.
- **Linking stage (.o, .a → .exe)**
 - Links the object code into an executable.
 - May involve one or more object code files.
 - **Function calls** are matched up with their definitions, and the compiler checks to make sure it has one, and only one, definition for every function.
 - The end result of linking is usually an executable.

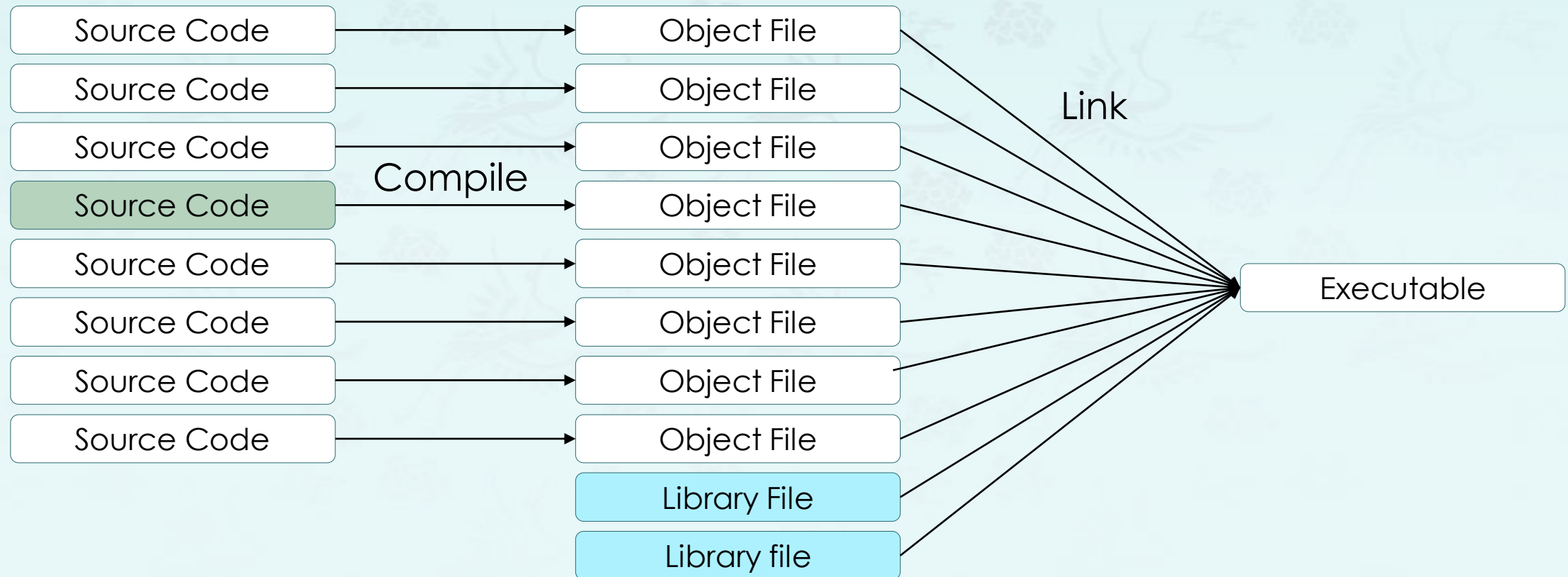
Compiling options

- **-g** - turn on debugging (so GDB gives more friendly output)
 - **-Wall** - turns on most warnings
 - **-O** or **-O2** - turn on optimizations
 - **-o <name>** - name of the output file
 - **-c** - output an object file (**.o**)
 - **-I<include path>** - specify an **include** directory
 - **-L<library path>** - specify a **lib** directory
 - **-l<library>** - link with library **lib<library>.a**
-
- Use **-Ldir** option such that linker looks for library files in **dir** folder.
Use **-llibrary** such that linker searches the library named **library**.

Build Process

Addressing the build process efficiency:

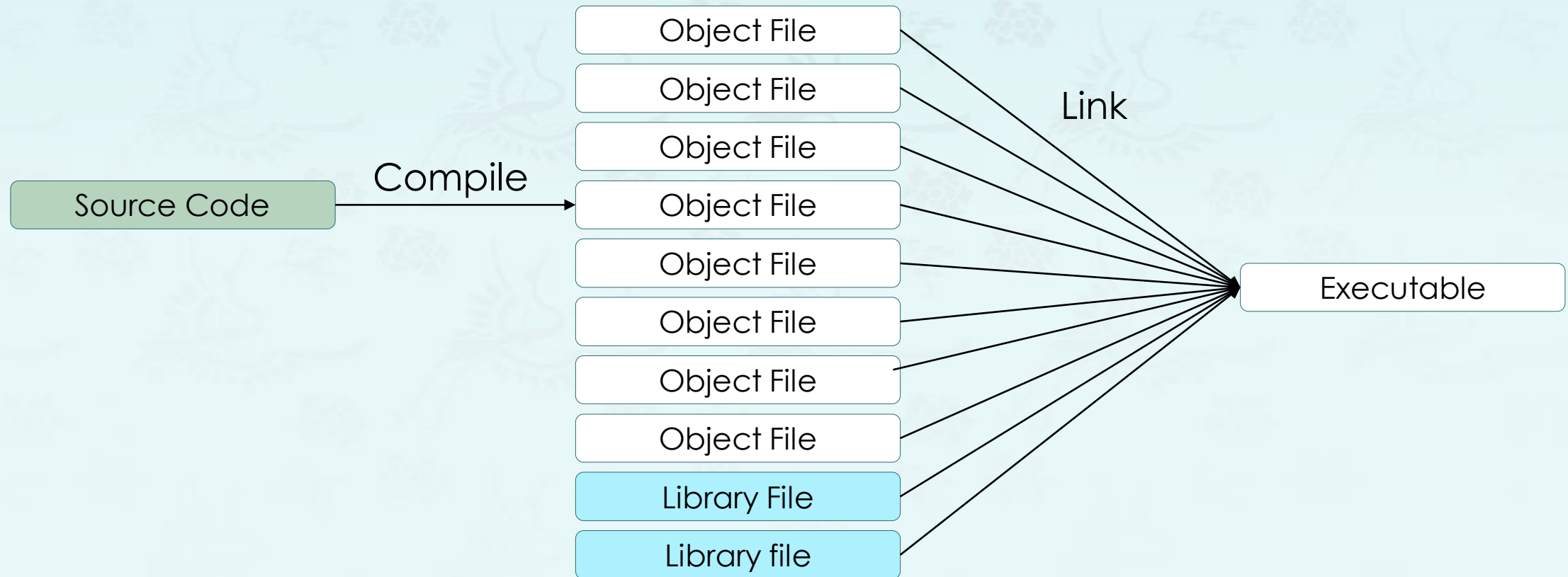
- In this case we highlight the situation where ONE of those files has changed since the previous build. In building the naive brute-force way, ALL source code is recompiled, even those source code files that have not changed.



Build Process

Addressing the build process efficiency:

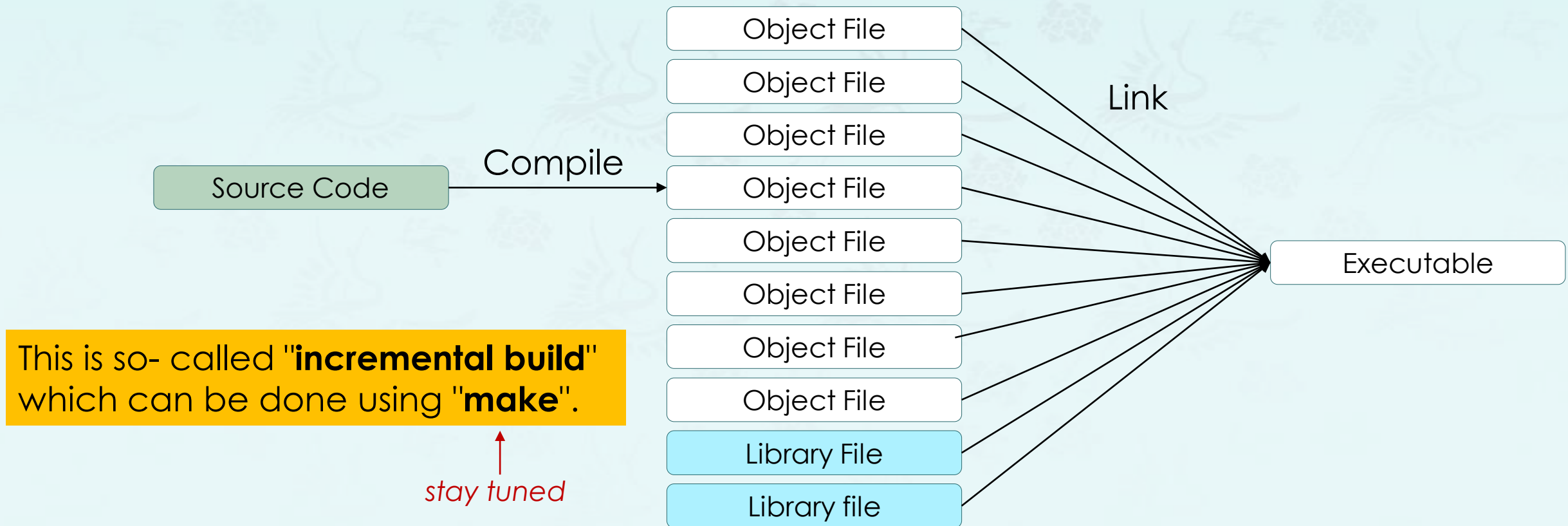
- Only one source file has changed since the last build. This is the only file re-compiled on the following build. The build otherwise uses the unchanged object files from the previous build, shortening the overall build time.



Build Process

Addressing the build process efficiency:

- Only one source file has changed since the last build. This is the only file re-compiled on the following build. The build otherwise uses the unchanged object files from the previous build, shortening the overall build time.



Creating a Library Archive

An archive in C/C++ is a file that bundles a set of object files into a single file.

- This file always follows the naming convention of starting with **lib** and ending with **.a**, e.g., `libsort.a` and `libnowic.a` in our library examples.
- An archive file can be built from object files using the **ar** command:

```
> ar crs libsort.a bubble.o insertion.o quick.o selection.o
```
- **ar** Options
 - c: Create an archive file
 - r: Insert the files member... into archive (with replacement).
 - s: Write an object-file index into the archive, change is made to the archive
 - t: Display contents of archive (show the list of .o files, use `nm ~.o` to see functions in ~.o)

Creating a Library Archive

- **ar** Examples:
 - `g++ -c nowic.cpp -I../include` // produces `nowic.o`
 - `ar crs libnowic.a nowic.o` // produces `libnowic.a` that includes `nowic.o`
 - `ar` // list all the options available
 - `ar t libnowic.a` // list `~.o` files archived
 - `ar x libnowic.a` // extract `~.o` files archived
 - `nm nowic.o` // list the actual function names in `.o` file
- You may refer to **`/nowic/UsingStaticLib.md`**.

The make utility

- Building a program from its source files can be a complicated and time-consuming operation. The commands are too long to be typed in manually every time. However, a straightforward shell script is seldom used for compiling a program, because it's too time-consuming to recompile all modules when only one of them has changed.
- However, it's too error-prone to allow a human to tell the computer which files need to be recompiled. Forgetting to recompile a file can mean hours of frustrating debugging. A reliable automatic tool is necessary for determining exactly which modules need recompilation.
- A standard tool for solving exactly this problem is called **make**. It relies either on its own built-in knowledge, or on a file called a **makefile (or Makefile)** that contains a detailed recipe for building the program.

References:

- <http://nuclear.mutantstargoat.com/articles/make/>
- <https://skandhurkat.com/post/makefile-dependencies/>
- <https://www.tuwlab.com/ece/27193> ← *one of the best, must-read*

The make utility

- You may need to install some packages.
(Install it as **admin privilege**. 관리자모드로 설치하십시오)
 - > `pacman -S base-devel` #install the build toolchain
 - > `pacman -Syu` #update msys2Alternatively,
 - > `pacman -S --needed base-devel mingw-w64-i686-toolchain mingw-w64-x86_64-toolchain`
- For macOS, use 'Homebrew' to install these kinds of packages in general.
 - <https://osxdaily.com/2018/03/07/how-install-homebrew-mac-os/>
 - <https://whitepaek.tistory.com/3>
- For Linux,
 - > `./configure`
 - > `make`
 - > `sudo make install`

The make utility – a simple example of makefile

- **Basic syntax for makefile:**

```
target: dependencies  
<tab>system command(s)
```

- **Example:**

Source files: quick.cpp, comparator.cpp, printlist.cpp
Executable: qsort.exe

- **makefile:**

The make utility – a simple example of makefile

- **Basic syntax for makefile:**

```
target: dependencies
<tab>system command(s)
```

- **Example:**

Source files: quick.cpp, comparator.cpp, printlist.cpp
Executable: qsort.exe

- **makefile:**

```
qsort: quick.o comparator.o printlist.o
    g++ quick.o comparator.o printlist.o -o qsort
quick.o: quick.cpp
    g++ -c quick.cpp -I../../include
comparator.o: comparator.cpp
    g++ -c comparator.cpp
printlist.o: printlist.cpp
    g++ -c printlist.cpp
clean:
    rm -f *.o qsort.exe qsort
```

```
makefile:14: ***분리기호가 빠졌음. 멈춤.
```

```
PS C:\GitHub\nowicx\labs\labmake> make
makefile:14: *** missing separator. Stop.
PS C:\GitHub\nowicx\labs\labmake>
```

The make utility – a simple example of makefile

- **Basic syntax for makefile:**

```
target: dependencies  
<tab>system command(s)
```

- **Example:**

Source files: quick.cpp, comparator.cpp, printlist.cpp
Executable: qsort.exe

- **makefile:**

```
qsort: quick.o comparator.o printlist.o  
    g++ quick.o comparator.o printlist.o -o qsort  
quick.o: quick.cpp  
    g++ -c quick.cpp -I../../include  
comparator.o: comparator.cpp  
    g++ -c comparator.cpp  
printlist.o: printlist.cpp  
    g++ -c printlist.cpp  
clean:  
<tab>rm -f *.o qsort.exe qsort
```

```
> make  
> make clean
```

The make utility – a simple example of makefile

- **Basic syntax for makefile:**

```
target: dependencies
<tab>system command(s)
```

- **Example:**

Source files: quick.cpp, comparator.cpp, printlist.cpp
Executable: qsort.exe

- **makefile:**

```
qsort: quick.o comparator.o printlist.o
    g++ quick.o comparator.o printlist.o -o qsort
quick.o: quick.cpp
    g++ -c quick.cpp -I../../include
comparator.o: comparator.cpp
    g++ -c comparator.cpp
printlist.o: printlist.cpp
    g++ -c printlist.cpp
clean:
    <tab>rm -f *.o qsort.exe qsort
```

```
PS C:\Github\nowicx\labs\labmake> make
g++ -c quick.cpp -I../../include
g++ -c comparator.cpp
g++ -c printlist.cpp
g++ quick.o comparator.o printlist.o -o qsort
PS C:\Github\nowicx\labs\labmake> 
```

```
> make
> make clean
```

The make utility – a simple example of makefile

■ Basic syntax for makefile:

```
target: dependencies
<tab>system command(s)
```

■ Example:

Source files: quick.cpp, comparator.cpp, printlist.cpp

Executable: qsort.exe

■ makefile:

```
qsort: quick.o comparator.o printlist.o
    g++ quick.o comparator.o printlist.o -o qsort
quick.o: quick.cpp
    g++ -c quick.cpp -I../../include
comparator.o: comparator.cpp
    g++ -c comparator.cpp
printlist.o: printlist.cpp
    g++ -c printlist.cpp
clean:
    <tab>rm -f *.o qsort.exe qsort
```

- Turn on main() in quick.cpp since it is the only one in the source files. Otherwise, you may get the following error message.

```
C:/msys64/mingw64/bin/./lib/gcc/x86_64-w64-mingw32/10.2.0/./././././x86_64-w64-mingw32/
bin/ld.exe: C:/msys64/mingw64/bin/./lib/gcc/x86_64-w64-mingw32/10.2.0/./././././x86_64-
w64-mingw32/lib/./lib/libmingw32.a(lib64_libmingw32_a-crt0_c.o): in function `main':
D:/mingwbuild/mingw-w64-crt-git/src/mingw-w64/mingw-w64-crt/crt/crt0_c.c:18: undefined ref
erence to `WinMain'
collect2.exe: error: ld returned 1 exit status
make: *** [makefile:14: qsort] 오류 1
```

```
> make
> make clean
```

The make utility – sort example

- Source files and their dependencies are listed:
 - **sortDriver.cpp**: printlist.cpp, nowic.h, sort.h, libnowic.a
 - comparator.cpp
 - printlist.cpp
 - bubble.cpp: comparator.cpp, sort.h
 - quick.cpp: comparator.cpp, sort.h
- **Lab: Build an executable file: sortDriver.exe**
 - Using a command line:

```
> g++ -std=c++11 -Wall sortDriver.cpp comparator.cpp printlist.cpp bubble.cpp \
    quick.cpp -I../..../include -L../..../lib -lnowic -o sortDriver
```
 - Create a **'make.1'** to do the same command shown above:

The make utility – make.1

```
sortDriver: sortDriver.o comparator.o printlist.o bubble.o quick.o
```

```
<tab> g++ sortDriver.o comparator.o printlist.o bubble.o quick.o \  
      -I../../include -L../../lib -lnowic -o sortDriver
```

```
sortDriver.o: sortDriver.cpp
```

```
g++ -c -std=c++11 -Wall sortDriver.cpp -I../../include
```

```
comparator.o: comparator.cpp
```

```
g++ -c -std=c++11 -Wall comparator.cpp
```

```
printlist.o: printlist.cpp
```

```
g++ -c -std=c++11 -Wall printlist.cpp
```

```
bubble.o: bubble.cpp
```

```
g++ -c -std=c++11 -Wall bubble.cpp -I../../include
```

```
quick.o: quick.cpp
```

```
g++ -c -std=c++11 -Wall quick.cpp -I../../include
```

```
clean:
```

```
<tab> rm -f *.o sortDriver.exe sortDriver
```

- Turn off main() in all other files, if any, since main() in sortDriver.cpp will be used.

```
> make -f make.1  
> make clean
```

The make utility – make.1

```
sortDriver: sortDriver.o comparator.o printlist.o bubble.o quick.o
```

```
<tab> g++ sortDriver.o comparator.o printlist.o bubble.o quick.o \  
      -I../../include -L../../lib -lnowic -o sortDriver
```

```
sortDriver.o: sortDriver.cpp
```

```
g++ -c -std=c++11 -Wall sortDriver.cpp -I../../include
```

```
comparator.o: comparator.cpp
```

```
g++ -c -std=c++11 -Wall comparator.cpp
```

```
printlist.o: printlist.cpp
```

```
g++ -c -std=c++11 -Wall printlist.cpp
```

```
bubble.o: bubble.cpp
```

```
g++ -c -std=c++11 -Wall bubble.cpp -I../../include
```

```
quick.o: quick.cpp
```

```
g++ -c -std=c++11 -Wall quick.cpp -I../../include
```

```
clean:
```

```
<tab> rm -f *.o sortDriver.exe sortDriver
```

```
# Using Rules
```

```
INCDIR = ../../include
```

```
SRCS = sortDriver.cpp comparator.cpp bubble.cpp ...
```

```
OBJS = $(SRCS:.cpp=.o)
```

```
TARGET = sortDriver
```

```
$(TARGET): $(OBJS)
```

```
g++ -I$(INCDIR) $(SRCS) -o $(TARGET)
```

```
> make -f make.1
```

```
> make clean
```

The make utility – make.1

```
sortDriver: sortDriver.o comparator.o printlist.o bubble.o quick.o
```

```
<tab> g++ sortDriver.o comparator.o printlist.o bubble.o quick.o \  
      -I../../include -L../../lib -lnowic -o sortDriver
```

```
sortDriver.o: sortDriver.cpp
```

```
g++ -c -std=c++11 -Wall sortDriver.cpp -I../../include
```

```
comparator.o: comparator.cpp
```

```
g++ -c -std=c++11 -Wall comparator.cpp
```

```
printlist.o: printlist.cpp
```

```
g++ -c -std=c++11 -Wall printlist.cpp
```

```
bubble.o: bubble.cpp
```

```
g++ -c -std=c++11 -Wall
```

```
quick.o: quick.cpp
```

```
g++ -c -std=c++11 -Wall
```

```
clean:
```

```
<tab> rm -f *.o sortDriver.
```

```
# Using Rules
```

```
INCDIR = ../../include
```

```
SRCS = sortDriver.cpp comparator.cpp bubble.cpp ...
```

```
OBJS = $(SRCS:.cpp=.o)
```

```
TARGET = sortDriver
```

```
PS C:\GitHub\nowicx\labs\lab7build> make -f make.1
```

```
g++ -c -std=c++11 -Wall sortDriver.cpp -I../../include
```

```
g++ -c -std=c++11 -Wall comparator.cpp
```

```
g++ -c -std=c++11 -Wall printlist.cpp
```

```
g++ -c -std=c++11 -Wall bubble.cpp -I../../include
```

```
g++ -c -std=c++11 -Wall quick.cpp -I../../include
```

```
g++ sortDriver.o comparator.o printlist.o bubble.o quick.o -I../../include -L../../lib -lnowic -o sortDriver
```

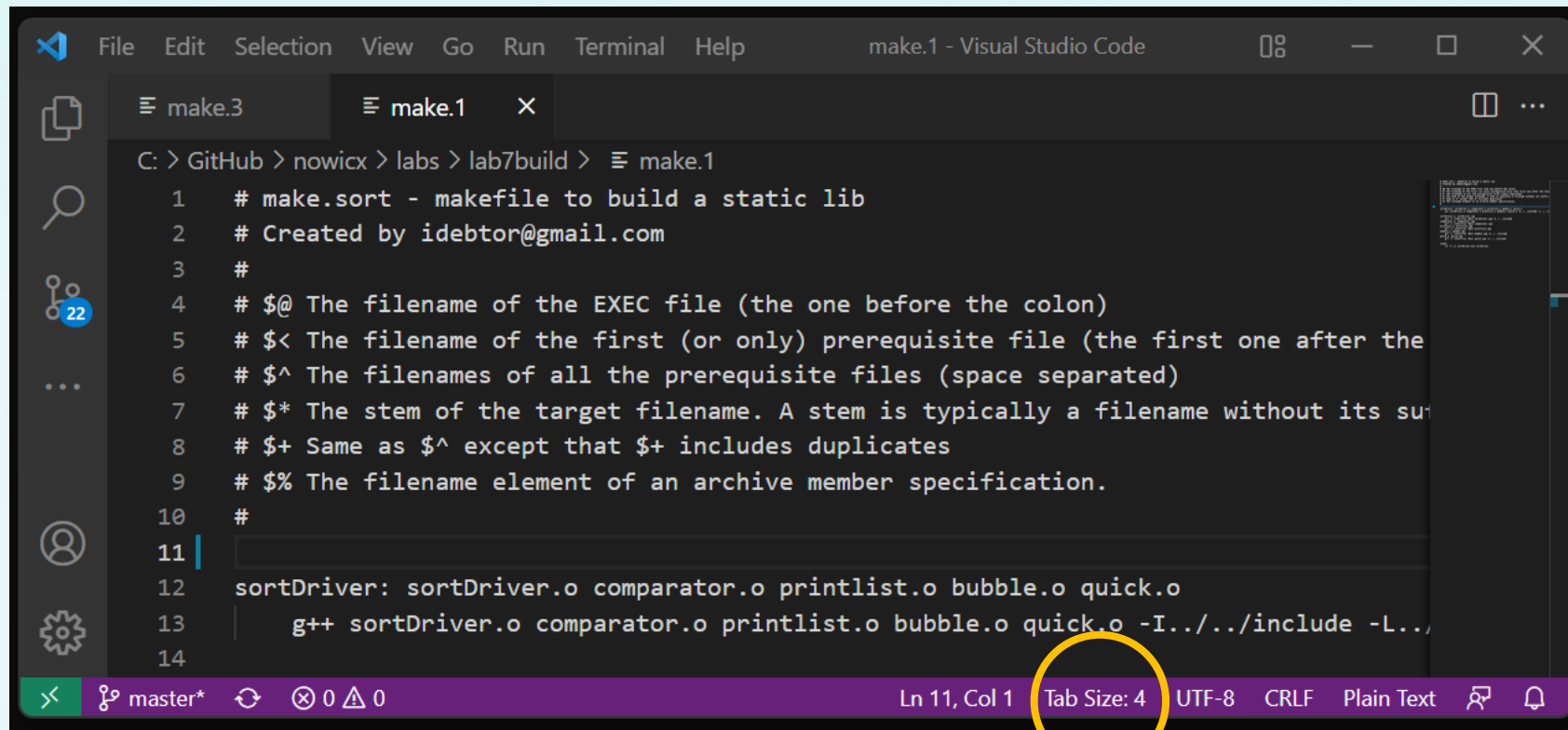
```
PS C:\GitHub\nowicx\labs\lab7build> █
```

```
> make -f make.1
```

```
> make clean
```

Error message: *** missing separator. Stop

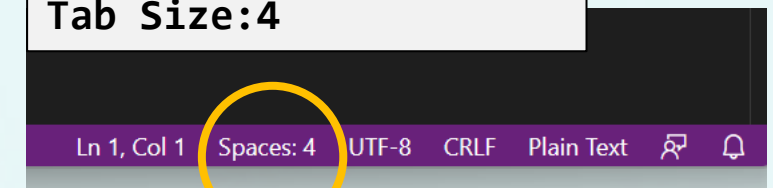
- This is one of the notorious error messages from "make"
- We must use <tab> instead of spaces when writing makefile. The problem is that some code editors including 'MS code' converts <tab> key input into a few spaces by default. Then you will see this error message.
 - Look for the option to change this default (from soft tab to hard tab) in your editor.



The screenshot shows the Visual Studio Code editor with a file named 'make.1' open. The file content is a Makefile with several comments and a rule. The status bar at the bottom indicates 'Ln 11, Col 1' and 'Tab Size: 4', which is circled in yellow. The error message '*** missing separator. Stop' is visible in the terminal output.

```
1 # make.sort - makefile to build a static lib
2 # Created by idebtor@gmail.com
3 #
4 # $@ The filename of the EXEC file (the one before the colon)
5 # $< The filename of the first (or only) prerequisite file (the first one after the
6 # $^ The filenames of all the prerequisite files (space separated)
7 # $* The stem of the target filename. A stem is typically a filename without its su
8 # $+ Same as $^ except that $+ includes duplicates
9 # $% The filename element of an archive member specification.
10 #
11
12 sortDriver: sortDriver.o comparator.o printlist.o bubble.o quick.o
13     g++ sortDriver.o comparator.o printlist.o bubble.o quick.o -I../include -L..
```

Click this and set it to
Indent Using Tabs
Tab Size:4



This close-up shows the status bar at the bottom of the editor, highlighting 'Spaces: 4' which is circled in yellow. Other status bar items include 'Ln 1, Col 1', 'UTF-8', 'CRLF', and 'Plain Text'.

The make utility – make.2

```
CC = g++
LIBDIR = ../../lib
INCDIR = ../../include
CCFLAGS = -Wall -std=c++11 -g
LDFLAGS = -L$(LIBDIR) -lnowic

SRCS = sortDriver.cpp comparator.cpp printlist.cpp bubble.cpp quick.cpp
OBS = $(SRCS:.cpp=.o)
EXE1 = sortDriver

$(EXE1): $(OBS)
    $(CC) -o $@ $^ $(LDFLAGS)
%.o: %.cpp
    $(CC) -c $(CCFLAGS) -I$(INCDIR) $<

.PHONY: all clean
all: $(EXE1)
clean:
    rm -f $(OBS) $(EXE1)
```

"For each target X, if X.cpp exists and is newer than X.o
(X.o does not exist), run the command below".

make automatic variables

`$@` the filename of the target

`$^` the filenames of all the prerequisites

`$<` the filenames of the first prerequisites

`$*` the stem of the target filename

`$+` Same as `$^` except that `$+` includes duplicates

The make utility – make.2

```
CC = g++
LIBDIR = ../../lib
INCDIR = ../../include
CCFLAGS = -Wall -std=c++11 -g
LDFLAGS = -L$(LIBDIR) -lnowic

SRCS = sortDriver.cpp comparator.cpp printlist.cpp bubble.cpp quick.cpp
OBS = $(SRCS:.cpp=.o)
EXE1 = sortDriver

$(EXE1): $(OBS)
    $(CC) -o $@ $^ $(LDFLAGS)
%.o: %.cpp
    $(CC) -c $(CCFLAGS) -I$(INCDIR) $<

.PHONY: all clean
all: $(EXE1)
clean:
    rm -f $(OBS) $(EXE1)
```

"For each target X, if X.cpp exists and is newer than X.o
(X.o does not exist), run the command below".

```
$(EXE1): $(OBS)
    $(CC) $(CCFLAGS) -I$(INCDIR) -o $@ $^ $(LDFLAGS)
```

make automatic variables

\$@ the filename of the target
\$^ the filenames of all the prerequisites
\$< the filenames of the first prerequisites
\$* the stem of the target filename
\$+ Same as \$^ except that \$+ includes duplicates

The make utility – make.2

```
CC = g++
LIBDIR = ../../lib
INCDIR = ../../include
CCFLAGS = -Wall -std=c++11 -g
LDFLAGS = -L$(LIBDIR) -lnowic

SRCS = sortDriver.cpp comparator.cpp printlist.cpp bubble.cpp quick.cpp
OBS = $(SRCS:.cpp=.o)
EXE1 = sortDriver

$(EXE1): $(OBS)
    $(CC) -o $@ $^ $(LDFLAGS)

%.o: %.cpp
    $(CC) -c $(CCFLAGS) -I$INCDIR $< -o $@

.PHONY: all clean
all: $(EXE1)
clean:
    rm -f $(OBS) $(EXE1)
```

```
> make -f make.2
> make clean
```

```
PS C:\Github\nowicx\labs\labmake> make -f make.2
g++ -c -Wall -std=c++11 -g -I../../include -o sortDriver.o sortDriver.cpp
g++ -c -Wall -std=c++11 -g -I../../include -o comparator.o comparator.cpp
g++ -c -Wall -std=c++11 -g -I../../include -o printlist.o printlist.cpp
g++ -c -Wall -std=c++11 -g -I../../include -o bubble.o bubble.cpp
g++ -c -Wall -std=c++11 -g -I../../include -o quick.o quick.cpp
g++ -o sortDriver sortDriver.o comparator.o printlist.o bubble.o quick.o -L../../lib -lnowic
PS C:\Github\nowicx\labs\labmake>
```

Build a static library(libsort.a)

- Using a command line
 - Build `libsort.a`, and place its copy to `../..lib` folder.
 - Assume that we have all source files at the current folder and `../..include/sort.h`.

```
> g++ -c -std=c++11 -Wall comparator.cpp
> g++ -c -std=c++11 -Wall printlist.cpp
> g++ -c -std=c++11 -Wall bubble.cpp -I../..include
> g++ -c -std=c++11 -Wall quick.cpp -I../..include
>
> ar cru libsort.a comparator.o printlist.o bubble.o quick.o
> ar t libsort.a
```

- Using a makefile

Build a static library(libsort.a) – make.libsort

```
CC = g++
LIBDIR = ../../lib
INCDIR = ../../include
CCFLAGS = -Wall -std=c++11 -g
LDFLAGS = -L$(LIBDIR)

SRCS = comparator.cpp printlist.cpp bubble.cpp quick.cpp
OBJS = $(SRCS:.cpp=.o)
ARCH = libsort.a

$(ARCH): $(OBJS)
    ar cru $@ $+
    ranlib $@

%.o: %.cpp
    $(CC) -c $(CCFLAGS) $<

.PHONY: all install clean
all: $(ARCH)
clean:
    rm -f $(OBJS)
install:
    cp -v $(ARCH) ../../nowic/lib
```

```
> make -f make.libsort
> ar t libsort.a
```

"For each target X, if X.cpp exists and is newer than X.o (X.o does not exist), run the command below".

```
make automatic variables
$@  the filename of the target
$^  the filenames of all the prerequisites
$<  the filenames of the first prerequisites
$*  the stem of the target filename
$+  Same as $^ except that $+ includes duplicates
```

Build a static library(libsort.a) – make.libsort

```
CC = g++
LIBDIR = ../../lib
INCDIR = ../../include
CCFLAGS = -Wall -std=c++11 -g
LDFLAGS = -L$(LIBDIR)

SRCS = comparator.cpp printlist.cpp bubble.cpp quick.cpp
OBS = $(SRCS:.cpp=.o)
ARCH = libsort.a

$(ARCH): $(OBS)
    ar cru $@ $+
    ranlib $@

%.o: %.cpp
    $(CC) -c $(CCFLAGS) $<

.PHONY: all install clean
all: $(ARCH)
clean:
    rm -f $(OBS)
install:
    cp -v $(ARCH) ../../nowic/lib
```

```
> make -f make.libsort
> ar t libsort.a
```

```
PS C:\GitHub\nowicx\labs\labmake> make -f make.libsort
g++ -c -Wall -std=c++11 -g bubble.cpp
ar cru libsort.a comparator.o printlist.o bubble.o quick.o
ranlib libsort.a
PS C:\GitHub\nowicx\labs\labmake> █
```

Build an executable using a static lib

- Source files and their dependencies are listed:
 - `sortDriver.cpp`: `nowic.h`, `sort.h`, `libnowic.a`, `libsort.a`
- Build an executable file: `sortDriver.exe`
 - `g++ -Wall -std=c++11 sortDriver.cpp -I.././include -L.././lib -lsort -lnowic -o sortDriver`

Build an executable(sortDriver) using a static lib – make.3

```
CC = g++
LIBDIR = ../../lib
INCDIR = ../../include
CCFLAGS = -Wall -std=c++11 -g
LDFLAGS = -L$(LIBDIR) -lnowic -lsort

SRC1 = sortDriver.cpp
OBJ1 = $(SRC1:.cpp=.o)
EXE1 = sortDriver

$(EXE1): $(OBJ1)
    $(CC) -o $@ $^ $(LDFLAGS)
%.o: %.cpp
    $(CC) $(CCFLAGS) -c $< -I$(INCDIR)

.PHONY: all clean
all: $(EXE1)
clean:
<tab> rm -f $(OBJ1) $(EXE1)
```

> make -f make.3

```
$(EXE1): $(OBJS)
    $(CC) $(CCFLAGS) -I$(INCDIR) -o $@ $^ $(LDFLAGS)
```

make automatic variables

\$@ the filename of the target

\$^ the filenames of all the prerequisites

\$< the filenames of the first prerequisites

\$* the stem of the target filename

\$+ Same as \$^ except that \$+ includes duplicates

Build an executable(sortDriver) using a static lib – make.3

```
CC = g++
LIBDIR = ../../lib
INCDIR = ../../include
CCFLAGS = -Wall -std=c++11 -g
LDFLAGS = -L$(LIBDIR) -lnowic -lsort
```

```
SRC1 = sortDriver.cpp
OBJ1 = $(SRC1:.cpp=.o)
EXE1 = sortDriver
```

```
$(EXE1): $(OBJ1)
    $(CC) -o $@ $^ $(LDFLAGS)
%.o: %.cpp
    $(CC) $(CCFLAGS) -c $< -I$(INCDIR)

.PHONY: all clean
all: $(EXE1)
clean:
```

```
<tab> rm -f $(OBJ1) $(EXE1)
```

- Make sure that you have only one main() turned on in sortDriver.cpp.

```
> make -f make.3
```

```
PS C:\Github\nowicx\labs\labmake> make -f make.3
g++ -c -Wall -std=c++11 -g -I../../include -o sortDriver.o sortDriver.cpp
g++ -c -Wall -std=c++11 -g -I../../include -o comparator.o comparator.cpp
g++ -c -Wall -std=c++11 -g -I../../include -o printlist.o printlist.cpp
g++ -c -Wall -std=c++11 -g -I../../include -o bubble.o bubble.cpp
g++ -c -Wall -std=c++11 -g -I../../include -o quick.o quick.cpp
g++ -o sortDriver sortDriver.o comparator.o printlist.o bubble.o quick.o -L../../lib -lnowic
PS C:\Github\nowicx\labs\labmake>
```

C++ For C Coders 8

Build Process

Data Structures
C++ for C Coders

한동대학교 김영섭교수
idebtor@gmail.com

build process
compile & link
static library
make & makefile