Compiling Code on Linux

FORDHAM

CISC 5004 - Programming C++
Sam Kamens
Fall, 2022

Compile single file

```
$ cat hello_world.cpp
#include <iostream>
using namespace std;
int main()
{
    cout << "Hello World!" << endl;
    return 0;
}
$ g++ hello_world.cpp -o hello_world
$ ./hello_world
Hello World!
```

Compile Steps

1. Preprocessor
   - Reads the source file looking for **preprocessor directives** (lines beginning with '#')
     - `#include <iostream>` - preprocessor replaces this line with the entire contents of the `iostream` header file
       - Header file guards

```
#ifndef FILENAME_H
#define FILENAME_H

// Header file contents

#endif
```

   - Removes comments and extra white space

Compile Steps

2. Compiler
    - Processes each statement in order, translating to machine code
    - Reports errors and warnings, if any (and fails on error)
    - Writes an **object file** containing the binary code

Compile Steps

3. Linker
    - Combines object file(s) with system libraries
        - e.g. the code implementing cout
    - Writes the actual **executable program** (`hello_world`)
    - Sometimes you need to tell the compiler about additional libraries you want to link with your program.
        - `g++ file.cpp -o file -lm`
            - -lm says to link with the math library

Running your code

```
$ ./hello_world
Hello World!
```

- Prefix with `./` to run the command in the current directory
  - Unless your PATH environment variable contains `.`
    - `PATH=/bin:/usr/local/bin:$HOME/bin:.`

Single Step Compilation

- Compile multiple files into a program using a single command
  - `g++ my_program.cpp my_class.cpp -o my_program`
  - Pro: Simple command line
  - Cons:
    - If any file changes, all must be recompiled
      - This can take a long time when many files are involved
    - Any user of `my_class` will need to have access to the source code

Modular Compilation

- Compile each source file (`.cpp`) to an object file (`.o`)
  - Multiple `.o` files can also be combined into **libraries** (`.a`)
- Link object files together to create the program

```
$g++ -c my_program.cpp
$g++ -c my_class.cpp
$g++ my_program.o my_class.o -o my_program
```

- See zyBook

make

- How can you make sure the right files get recompiled when they need to be?
- `make`
  - Command that can build your project based on rules
  - Rules defined in a makefile (usually `Makefile`)
- Makefile format

```
target : prerequisite1 prerequisite2 ... prerequisiteN
    command1
    command2
    ...
    commandN
```

- **Important**: Lines with commands must start with a Tab character (not spaces)

Makefile example

```
my_program : my_program.o my_class.o
    g++ my_program.o my_class.o -o my_program

my_program.o : my_program.cpp my_class.h
    g++ -Wall -c my_program.cpp

my_class.o : my_class.cpp my_class.h
    g++ -Wall -c my_class.cpp

clean :
    rm *.o my_program
```

See zyBook