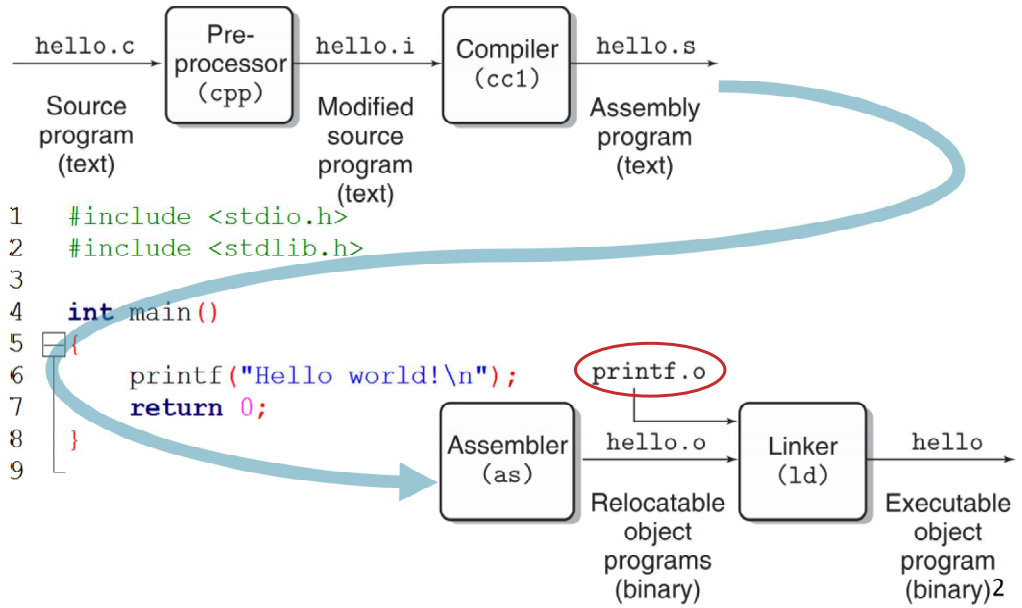
w02-Lab

C Compiling and Linking

Assembled for 204112

by Kittipitch Kuptavanich

Building a C Application (Revisited)



Steps of C Compilation

- **Preprocessing phase** - The **preprocessor** (cpp) modifies the original C program according to directives that begin with the # character.
  - For example, the `#include <stdio.h>` command in line 1 of `hello.c` tells the preprocessor to read the contents of the system header file `stdio.h` and insert it directly into the program text. The result is another C program, typically with the .i suffix: `hello.i`
- **Compilation phase** - The **compiler** (cc1) translates the text file `hello.i` into the text file `hello.s`, which contains an assembly-language program. Each statement in an assembly-language program

Steps of C Compilation [2]

- **Assembly phase** - Next, the **assembler** (as) translates `hello.s` into machine- language instructions, packages them in a form known as a relocatable object program, and stores the result in the object file `hello.o`
- **Linking phase** - Notice that our hello program calls the `printf` function, which is part of the standard C library provided by every C compiler. The `printf` function resides in a separate precompiled object file called `printf.o`, which must somehow be merged with our `hello.o` program. The **linker** (ld) handles this merging. The result is the hello file, which is an executable object file (or simply executable) : `hello.exe` that is ready to be loaded into memory and executed by the system.

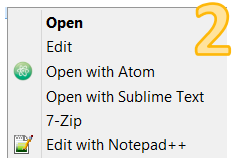
Compiling C using gcc

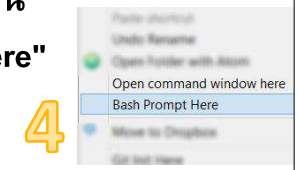
- GCC ย่อมาจาก GNU (g-noo) Compiler Collection (originally: GNU C Compiler) เป็นชุด compiler ของภาษาต่าง ๆ เช่น C/C++, Ada, Java, Pascal ….
- GCC เป็น compiler ที่เกิดและพัฒนาในระบบปฏิบัติการตระกูล unix
- ระบบปฏิบัติการ windows เราสามารถใช้ GCC โดยการ install MinGW GCC หรือ Cygwin GCC
- Cygwin is a Unix-like environment and command-line interface for Microsoft Windows

This course

Tutorial 1: Creating Files - method 1

1. สร้างไฟล์เปล่าลงใน folder ที่ต้องการชื่อ hello.c
   1. Click ขวา และเลือก New -> Text Document
   2. Rename File ทั้งชื่อและนามสกุลให้เป็น hello.c
2. แล้ว click ขวาเลือก "Open with Sublime Text" หรือ "Edit with Notepad++"
3. พิมพ์ source code ลงไปตามปกติ
4. เปิด Cygwin bash prompt ขึ้นใน folder เดียวกันด้วยการ click ขวาแล้วเลือก "Bash Prompt Here"





Compiling C using gcc [2]

- ในการ compile program ภาษา C ด้วย GCC เราใช้คำสั่ง gcc

```
$ gcc -Wall hello.c -o hello
```

- ในการ run program ที่เป็นผลลัพธ์ ทำได้โดย

```
$ ./hello
```


Compiling C using gcc [3]

```
$ gcc -Wall hello.c -o hello
```

flags (or command line flags)

- ในการทำงานบน command line environment การกำหนด flag ในการเรียกใช้คำสั่งต่าง ๆ (gcc ในกรณีนี้) เป็นวิธีพื้นฐานในการระบุ ตัวเลือก (option) ในการใช้คำสั่ง
- การ compile program ภาษา C ด้วยคำสั่ง gcc นั้น สามารถทำได้โดยไม่จำเป็นต้องมีการระบุ flag

```
$ gcc hello.c
```

Default output file ชื่อ_____

from Carnegie Mellon University's 15213 course recitation slide by Marjorie Carlson

gcc Command Line Flag

- **gcc does not requires these flags, but they encourage people to write better C code.**

Useful Flags	
-Wall	Enables all construction warnings
-Wextra	Enables even more warnings not enabled by Wall
-Werror	Treat all warnings as Errors
-ansi	Compiles code according to 1989 C standards
-g	Produces debug information (GDB uses this information)
-O1	Optimize
-O2	Optimize even more
-o filename	Names output binary file "filename"

More at: https://gcc.gnu.org/onlinedocs/gcc/Overall-Options.html#Overall-Options

gcc Compilation Steps

ในการ compile source code ภาษา C ด้วยคำสั่ง gcc ประกอบด้วยขั้นตอนย่อย 4 ขั้นตอน

ตัวอย่างเช่น "gcc -o hello.exe hello.c" ประกอบด้วยขั้นตอนดังนี้

STEP1: Pre-processing:

```
$ cpp hello.c > hello.i
```

STEP2: Compilation:

```
$ gcc -S hello.i
```

(The -S option specifies to produce assembly code,)

STEP3: Assembly:

```
$ as -o hello.o hello.s
```

STEP4: Linker:

```
$ ld -o hello.exe hello.o ...libraries...
```

เราสามารถดูขั้นตอนย่อย ๆ ในการ compile โดยการใช้ -v (verbose) flag.

For example,

```
$ gcc -v hello.c -o hello.exe
```


Compiling Multiple Files

- **ในกรณีที่ project มี source code หลายไฟล์**

factorial.c

```c
#include "factorial.h"

int factorial(int x)
{
  int result = 1;

  for (; x >= 1; x--) {
    result = result * x;
  }

  return result;
}
```

factorial.h

```c
#include <stdio.h>

int factorial(int x);
```

C header file content
- Directives & Macro ต่าง ๆ เช่น
  - #include
  - #define
- Function prototype

main.c

```c
#include "factorial.h"

int main ()
{

  int x;
  printf("Please input x: ");
  scanf("%d",&x);

  int ans = factorial(x);
  printf("ans= %d\n",ans);

  return 0;

}
```


Tutorial 2: Creating Files - method 2

1. เปิด Cygwin bash prompt ขึ้นใน folder ที่ต้องการ ด้วยการ click ขวาแล้วเลือก "Bash Prompt Here"
2. สร้าง folder project ด้วยคำสั่ง

```
$ mkdir factorial
```

3. Change directory เข้าไปใน folder ดังกล่าว ด้วยคำสั่ง

```
$ cd factorial
```

4. สร้างไฟล์เปล่า factorial.c factoria.h main.c

```
$ touch factorial.c factorial.h main.c
```

5. Edit file ด้วย Sublime Text หรือ Notepad++

Compiling Multiple Files [2]

- การ **compile** จะต้อง **compile** ทุกไฟล์ที่เกี่ยวข้อง

```
$ gcc -Wall main.c factorial.c -o factorial
```

ข้อเสีย

- ต้องพิมพ์คำสั่งในการ **compile** ใหม่ทุกครั้ง
- หาก **project** มีขนาดใหญ่ (ประกอบด้วยไฟล์ **source code** จำนวนมาก) และมีการแก้ไข **file** เพียงไม่กี่ไฟล์ ก็จะต้อง **compile** ทุกไฟล์ใหม่

Solution: Makefiles

from Carnegie Mellon University's 15213 course recitation slide by Marjorie Carlson

make and Makefiles

- **Makefile** จะมีข้อมูล **compiler** ที่ใช้ และ **flag** ที่ใช้ในการ **compile**, ไฟล์ **source code** ที่ต้องการ **compile**, ชื่อไฟล์ **output**
- **Makefile** ใช้แก้ปัญหาที่กล่าวมาข้างต้น โดยสามารถที่จะ **compile source code** ใหม่ เฉพาะส่วนที่มีการเปลี่ยนแปลงและนำ **objet file** (`.o`) มา **link** เข้ากับส่วนที่ไม่มีความเปลี่ยนแปลง
- โดย **Makefile** จะแยกการ **compile** (`.c -> .o`) ออกจากการ **link** (`.o -> executable`)
- ทั้งนี้การ **compile** โดยการใช้ **Makefile** จะทำผ่านคำสั่ง `make`

from Carnegie Mellon University's 15213 course recitation slide by Marjorie Carlson

C – Compiling: Makefiles

- **Makefiles consist of one or more rules in the following form.**

Makefile Rule Format	Makefile for "gcc foo.c bar.c baz.c –o myapp"

target … : *prerequisites* …
[TAB]*recipe*
[TAB]…
[TAB]…
	myapp: foo.o bar.o baz.o
	  gcc foo.o bar.o baz.o –o myapp
	foo.o: foo.c foo.h
	  gcc –c foo.c
	bar.o: bar.c bar.h
	  gcc –c bar.c
	baz.o: baz.c baz.h
	  gcc –c baz.c

- โดยเมื่อต้องการ **compile program** สามารถใช้คำสั่ง `make` ตามด้วย ชื่อ **target** ที่ต้องการสร้าง หากไม่ระบุ **default** คือ **target** แรกที่ปรากฏใน **Makefile**

```
$ make myapp
```


from Carnegie Mellon University's 15213 course recitation slide by Marjorie Carlson

C – Compiling: Makefiles [2]

```
factorial: main.o factorial.o
    gcc -Wall -o factorial main.o factorial.o

main.o: main.c factorial.h
    gcc -Wall -c main.c

factorial.o: factorial.c factorial.h
    gcc –Wall -c factorial.c
```

Makefile Rule Format

```
target … : prerequisites …
[TAB]recipe
[TAB]…
[TAB]…
```

Tutorial 3: Compiling with Makefile

1. เปิด Cygwin bash prompt ขึ้นใน project folder (factorial) ด้วยการ click ขวาแล้วเลือก "Bash Prompt Here"
2. สร้างไฟล์ Makefile ด้วยคำสั่ง

```
$ touch Makefile
```

3. Edit file ด้วย Sublime Text หรือ Notepad++
4. Compile project ด้วยคำสั่ง

```
$ make factorial
```

5. Run program ที่ได้จากการ compile ด้วยคำสั่ง

```
$ ./factorial
```


from Carnegie Mellon University's 15213 course recitation slide by Marjorie Carlson

C – Compiling: Makefiles [3]

- ตัวอักษรแรกของแต่ละ recipe จะต้องเป็น tab (`'\t'`)
- การพิจารณาว่า ไฟล์ใดมีความเกี่ยวข้องในลักษณะ dependency กับไฟล์ใดสามารถทำได้โดย
  - `gcc -MM foo.c` outputs foo's dependencies to the console.
  - `makedepend` adds dependencies to the Makefile for you, if you already have one. E.g., `foo.c bar.c baz.c` .

C – Compiling: Makefiles [4]

- เราสามารถใช้ variables (macros) ในการเขียน Makefile ได้ เช่น

```
GCC=gcc
FLAGS=-Wall
all: factorial
factorial: main.o factorial.o
    $(GCC) $(FLAGS) -o factorial main.o factorial.o

main.o: main.c factorial.h
    $(GCC) $(FLAGS) -c main.c

factorial.o: factorial.c factorial.h
    $(GCC) $(FLAGS) -c factorial.c

clean:
    rm main.o factorial.o factorial
```

variables

```
#-c = compile only
```

Note:
โดยทั่วไปใน Makefile จะมีการสร้าง `clean` target ไว้ เพื่อใช้ในการลบไฟล์ทุกไฟล์ที่ไม่ใช่ source code เพื่อใช้ในกรณีที่ต้องการให้มีการ compile ทุกไฟล์ใหม่ทั้ง project

Advanced

C – Compiling: Makefiles [5]

- สังเกตว่า ไฟล์ที่มีนามสกุล (Suffix) `.o` จะถูก compile จาก ไฟล์ `.c` ที่มีชื่อเดียวกัน
- เราสามารถใช้ ***pattern rule*** แทนใน Makefile ได้

```
GCC=gcc
FLAGS=-Wall
all: factorial
factorial: main.o factorial.o
    $(GCC) $(FLAGS) -o factorial main.o factorial.o
#-c = compile only
%.o : %.c factorial.h
      $(GCC) $(FLAGS) -c $< -o $@
clean:
    rm main.o factorial.o factorial
```

Automatic Variables

Advanced

C – Compiling: Makefiles [6]

Notable Automatic Variables

- $@
  - The file name of the target of the rule.
- $%
  - The target member name
- $<
  - The name of the first prerequisite
- $?
  - The names of all the prerequisites that are newer than the target, with spaces between them.
- $^
  - The names of all the prerequisites, with spaces between them.

BASIC UNIX COMMANDS

from Carnegie Mellon University's 15213 course recitation slide by Marjorie Carlson

Command Line Interface

- **Command Line Interface**
  - "Provides a means of communication between a user and a computer that is based solely on textual input and output."
- **Shell**
  - A shell is a program that reads and executes the commands entered on the command line. It provides "an interface between the user and the internal parts of the operating system (at the very core of which is the kernel)."
  - The original UNIX shell is `sh` (the Bourne shell).
  - Many other versions exist: `bash`, `csh`, etc.

from Carnegie Mellon University's 15213 course recitation slide by Marjorie Carlson

Unix – Beginner: Basic Commands

Moving Around		Note
`ls`	List directory contents	
`cd`	Change working directory	
`pwd`	Display present working directory	
`ln`	Make links between files/directories	
`mkdir`	Make directories	
Searching Files and File Content		
`find`	Search for files in a directory hierarchy	
`grep`	Print lines matching a pattern	
Other Important Commands		
`echo`	Display a line of text	
`exit`	Cause the shell to exit	
`history`	Display the command history list	
`touch`	Change file timestamps (creates empty file, if nonexistent)	

from Carnegie Mellon University's 15213 course recitation slide by Marjorie Carlson

Unix – Beginner: Basic Commands

Manipulating Files		Note
`mv`	Move (rename) files	
`cp`	Copy files (and directories with "-r")	
`rm`	Remove files (or directories with "-r")	
`cat`	Concatenate and print files	
`chmod`	Change file permission bits	
Looking Up Commands		
`man`	Interface to online reference manuals	
`which`	Shows the full path of shell commands	
`locate`	Find files by name	
Managing Processes		
`ps`	Report current processes status	
`kill`	Terminate a process	
`jobs`	Report current shell's job status	

from Carnegie Mellon University's 15213 course recitation slide by Marjorie Carlson

`man` page Section

- คำสั่ง `man foo` แสดง **manual page** (วิธีใช้) คำสั่ง `foo`, โดยที่ `foo` สามารถเป็นได้ทั้ง **any user command, system call, C library function, etc.**
- ในบางกรณี **program, command หรือ function** ใด อาจมีชื่อซ้ำได้ ดังนั้น เราจำเป็นต้อง ระบุ **section ของ manual page** ของ คำสั่ง / **function** ที่ต้องการ
  - **(e.g., `man 3 printf` gets you the man page for the C library function `printf`, not the Unix command `printf`).**
- **How do you know which section you want?**
  - `whatis foo`
  - `man -f foo`
  - `man -k foo` **lists all man pages mentioning "foo."**

from Carnegie Mellon University's 15213 course recitation slide by Marjorie Carlson

Using the `rm` Command

- `rm ./filename` **– deletes file "filename" in current dir.**
- `rm ./*ame` **– deletes all files in the current directory that end in `"ame"`**
- `rm -r ./*` **– deletes all files and directories inside the current directory.**
- `rm -r ./directory` **– deletes all files inside the given directory and the directory itself.**

from Carnegie Mellon University's 15213 course recitation slide by Marjorie Carlson

Pipes & Redirects

- **A pipe (`|`) between two commands sends the first command's output to the second command as its input.**
- **A redirect (`<` or `>`) does basically the same thing, but with a file rather than a process as the input or output.**

Option 1: Pipes

```
$ find * | grep name
```

Option 2: Redirects

```
$ isMagicSQuare < input.txt > output.txt
```

Reference

- ***Computer Systems: A Programmer's Perspective*** **(2nd Edition) by Bryan and O'Hallaron**
- **https://www3.ntu.edu.sg/home/ehchua/programming/cpp/gcc_make.html**
- **https://gobyexample.com/command-line-flags**
- **https://gcc.gnu.org/onlinedocs/gcc/Overall-Options.html#Overall-Options**
- **http://www.thegeekstuff.com/2012/10/gcc-compiler-options/**
- **http://tumrai.com/คำสั่งยูนิกส์_Unix_Command_ผ่าน_SSH_Secure_Shell_เบื้องต้น**