w01-Lab

Getting Started with C Compiling

Assembled for 204112
Semester 2 - 2015
by Kittipitch Kuptavanich

Origins

- ภาษา C เป็นภาษาที่ถูกสร้างขึ้นมาเพื่อใช้เขียนระบบปฏิบัติการ ***Unix*** โดย **Dennis Ritchie** จาก **Bell Laboratories** ในช่วงปี **1969 - 1973**
- ในสมัยนั้น ระบบปฏิบัติการ หรือโปรแกรมระบบ **(System Program)** อื่น ๆ จะใช้ภาษา **Assembly** เป็นหลัก
  - โดยคอมพิวเตอร์แต่ละเครื่องมีระบบปฏิบัติการที่เขียนเพื่อขึ้นใช้เฉพาะเครื่องกับเครื่องนั้น ๆ
  - ไม่สามารถ **reuse code** ได้

```
1  section     .text
2  global      _start
3
4  _start:
5
6      mov     edx,len
7      mov     ecx,msg
8      mov     ebx,1
9      mov     eax,4
10     int     0x80
11
12     mov     eax,1
13     int     0x80
14
15 section     .data
16
17 msg     db  'Hello, world!',0xa
18 len     equ $ - msg
19
```


Assembly Language (Recap)

- ภาษาเครื่องเป็นชุดคำสั่งที่อยู่ในรูปของ*เลขฐานสอง* ซึ่งเป็นคำสั่งที่เครื่องคอมพิวเตอร์เข้าใจได้โดยที่ไม่ต้องมีตัวแปลภาษา
- **Assembly** เป็นภาษาที่พัฒนาต่อมาจากภาษาเครื่อง **(Machine Language - Binary Code)** จึงมีความใกล้เคียงกับภาษาเครื่องมาก แต่ยังต้องการตัวแปลภาษา

Assembler: ใช้สำหรับแปลภาษา **Assembly** ไปเป็นภาษาเครื่อง

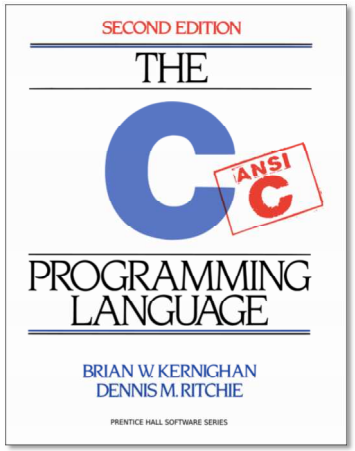
Origins

- **"Hello world" in C**

```c
#include <stdio.h>

int main()
{
    printf("hello, world\n");
    return 0;
}
```

- มาตรฐาน ภาษา **C** และ **Library** ของภาษาถูกรวบรวมไว้ในหนังสือที่เขียนโดย **Brian Kernighan and Dennis Ritchie (aka K&R – The C Bible)**

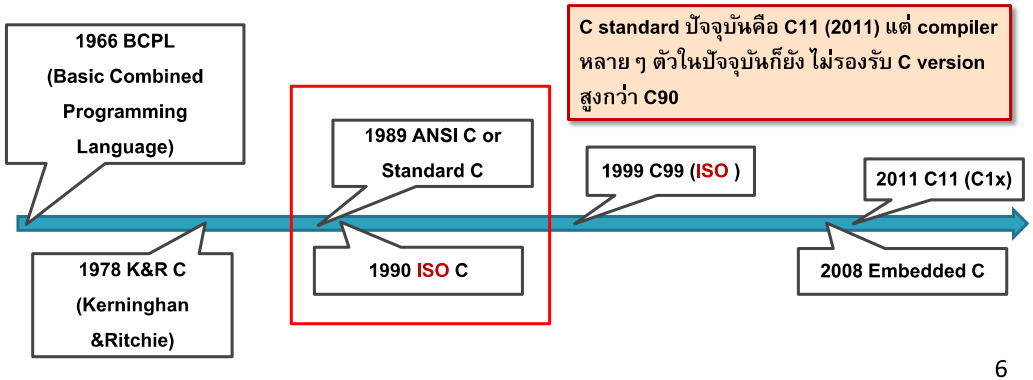


Origins

- ภาษา C ประสบความสำเร็จ และได้รับความนิยมอย่างมาก จากหลายปัจจัยได้แก่
  - C was closely tied with the Unix operating system.
  - C is a small, simple language
  - C was designed for a practical purpose.
  - C is the language of choice for system-level programming, (C++, Java - Application Level Programming)

C Version History

- หลังจากนั้น American National Standards Institute ได้ออกมาตรฐานภาษา C (ANSI C) ภายในปี 1989
- หน้าที่ในการออกมาตรฐานภาษา C ในปัจจุบันอยู่ภายใต้ International Organization for Standardization (ISO)



Tutorial 1: hello.c

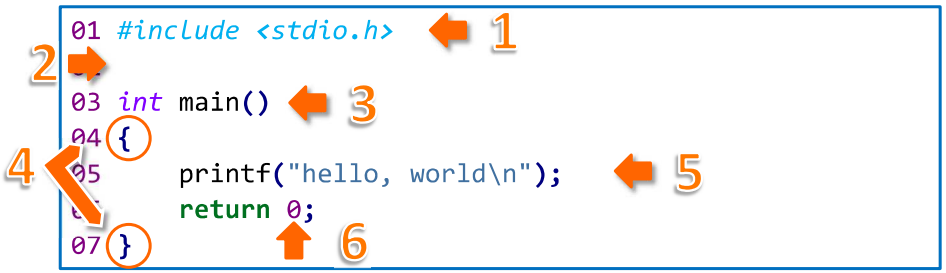
- ที่ bash prompt สร้าง file เปล่า (คำสั่ง touch) แล้วเปิดไฟล์มา edit ด้วย text editor ที่ถนัด (เช่น sublime text)

bash shell
```
$ touch hello.c
$ subl hello.c &
```

- แล้วพิมพ์ source code ดังแสดงลงไป

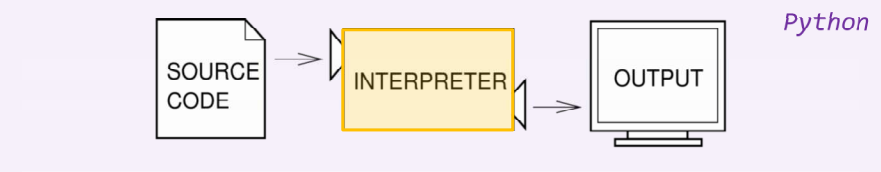
hello.c
```
01 #include <stdio.h>
02
03 int main()
04 {
05     printf("hello, world\n");
06     return 0;
07 }
```

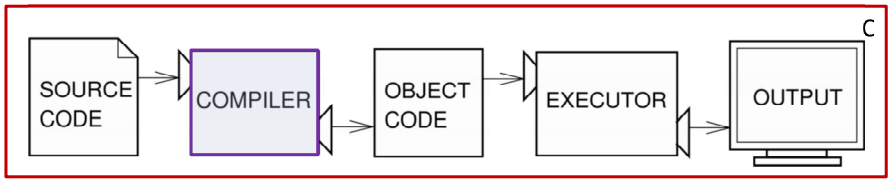

hello.c



1. Directives	
2. Blank Spaces	
3. Main Function	
4. Braces (Curly Brackets)	
5. Statement	
6. Function Return	

Interpreted vs Compiled



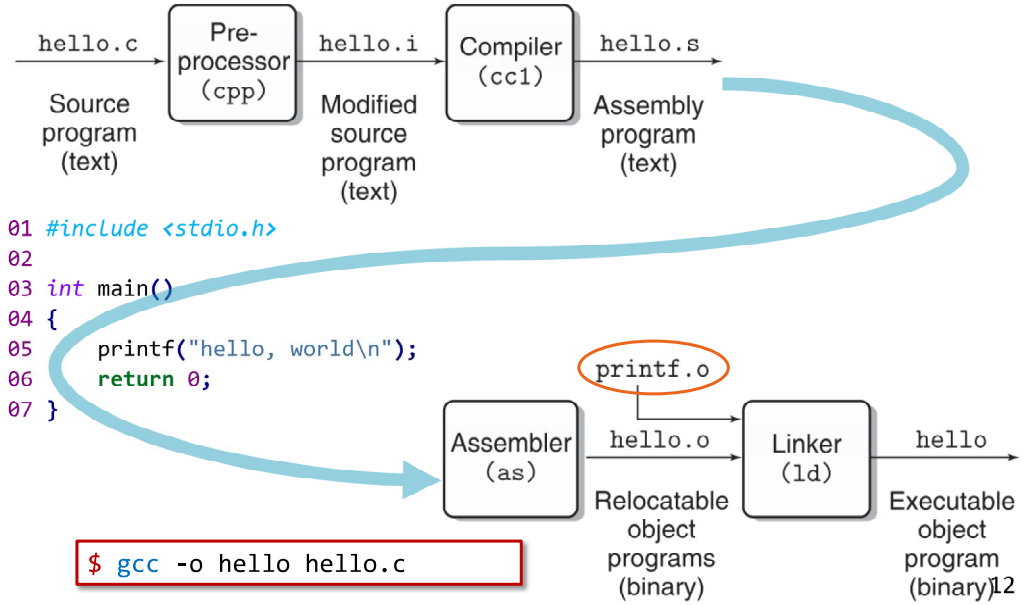


- ภาษา C จัดอยู่ในประเภท **Compiled Language**

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 (เช่น linux, bsd)**
- **บนระบบปฏิบัติการ windows เราสามารถใช้ GCC โดยการ install MinGW GCC หรือ Cygwin GCC**
- **Cygwin is a Unix-like environment and command-line interface for Microsoft Windows**

Building a C Application



Compiling C using gcc [2]

- **ในการ compile program ภาษา C ด้วย GCC เราใช้คำสั่ง gcc**

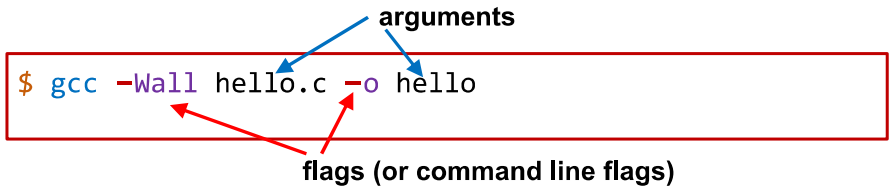
```
$ gcc -Wall hello.c -o hello
```

- **ในการ run program ที่เป็นผลลัพธ์ ทำได้โดย**

```
$ ./hello
```

เราสามารถใช้ ctrl + shift + b shortcut ในsublime text เพื่อ build และ run ได้เช่นกัน

Compiling C using gcc [3]



- ในการทำงานบน command line environment การกำหนด flag ในการเรียกใช้คำสั่งต่าง ๆ (gcc ในกรณีนี้) เป็นวิธีพื้นฐานในการระบุตัวเลือก (option) ในการใช้คำสั่ง flag จะขึ้นต้นด้วยอักขระ -
- 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.

Flags	Description	Notes
-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"	

COMPILING MULTIPLE FILES

Compiling Multiple Files

- ในกรณีที่ project มี source code หลายไฟล์

factorial.c

```
#include "factorial.h"

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

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

  return result;
}
```

factorial.h

```
#include <stdio.h>

int factorial(int x);
```

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

main.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

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
```

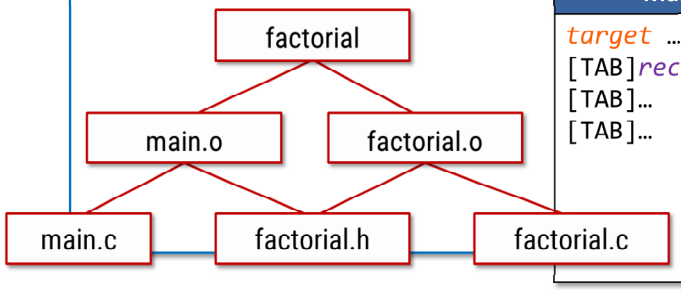
from Carnegie Mellon University's 15213 course recitation slide by Marjorie Carlson

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
```

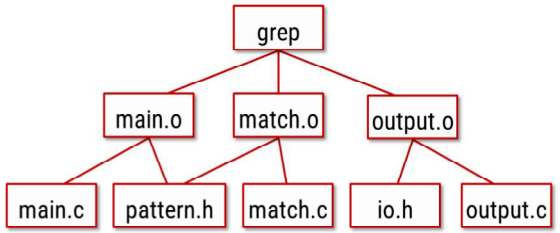

from Carnegie Mellon University's 15213 course recitation slide by Marjorie Carlson

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` **.**

Practice

- ให้เขียน **Makefile** จาก **Dependency Tree** ของคำสั่ง **grep** ดังแสดง




```
grep: _____.o _____.o _____.o
    gcc _____
_____: _____
    gcc _____
_____: _____
    gcc _____
_____: _____
    gcc _____
```

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.

References

- ***Computer Systems: A Programmer's Perspective*** **(2nd Edition) by Bryan and O'Hallaron**
- 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/