作業系統
Lab3- Makefile編寫

副標題

make 簡介

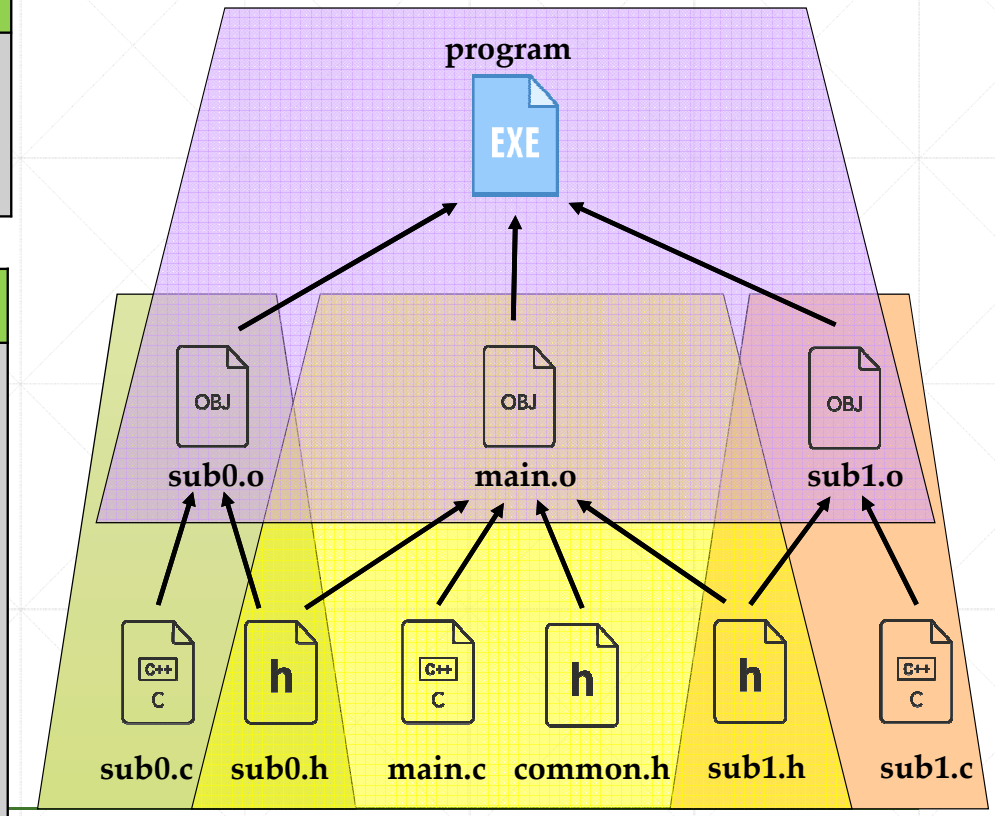
- 在軟體開發中，make 是一個工具程式，經由讀取一個叫做 Makefile 的檔案，自動編譯建構出執行檔、函式庫或核心模組。
- make會根據Makefile 裡面的目標 (target)、規則 (rule) 和檔案的修改時間進行判斷哪些需要重新編譯，可以省去大量重複編譯的時間，這在大型程式中尤為有用。

多檔程式專案編譯範例

common.h	sub0.h	sub1.h
/* common.h */ #define COUNT 100	void fun0(void);	void fun1(void);

main.c	sub0.c	sub1.c

/* main.c */
#include <stdio.h>
#include "common.h"
#include "sub0.h"
#include "sub1.h"
 	
int main()
{
printf("main\n");
fun0();
fun1();
return 0;
}
	#include <stdio.h>
	#include <sub0.h>
	
	void fun0(void) {
	printf("0000\n");
	return;
	}
		#include <stdio.h>
		#include <sub1.h>
		
		void fun1(void) {
		printf("1111\n");
		return;
		}



多檔程式專案範例

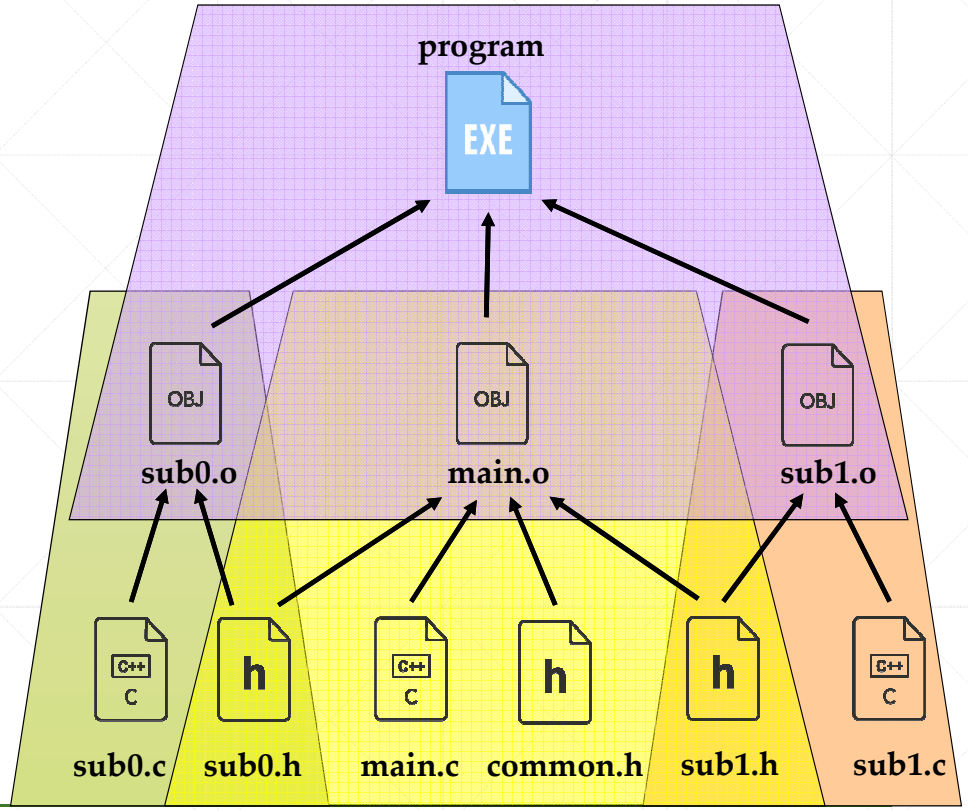
- 執行檔建構方式
  1. 先個別編譯所有 .c 檔產生 .o 檔
  2. 再由 .o 檔產生可執行檔
- gcc編譯命令下法1：

```
gcc -c main.c
gcc -c sub0.c
gcc -c sub1.c
gcc -o prorgam main.o sub0.o sub1.o
```

- gcc編譯命令下法2：

```
gcc -o prorgam main.c sub0.c sub1.c
```

- 若有任一原始檔被修改，下法1及2都會重新編譯所有.c檔案，再產生出執行檔。



Makefile的處理

- Makefile 規則
  - makefile 是由很多相依性項目（dependencies）和法則（rules）所組成。
  - 相依性項目，描述目標項目（target，要產生的檔案）和產生該檔案之相關的原始碼檔案。
  - 法則是說明如何根據相依性檔案，來建立目標項目。
  - make 命令利用 makefile，先決定依序建立哪些目標項目，再決定依序喚起哪些法則。
- 當我們輸入 make 命令，會執行以下內容：
  - 在當前目錄下按順序找尋文件名為 GNUmakefile、makefile 或 Makefile 的文件。
  - 在 makefile 文件中的找到第一個目標文件（target），並把這個文件作為最終的目標文件。
  - 如果沒找到或目標文件所依賴的文件，或修改時間要比目標文件新，則 make 將執行後面所定義的命令來生成這個文件，如此遞迴下去找到文件彼此的依賴關係，直到最終編譯出第一個目標文件。

make命令常用的選項及參數

- -j N：讓 make 在同一個時間執行 N 個命令，以加速編譯的時間。
- -k：讓 make 在遇到錯誤時，仍然繼續運行，不停止在第一個問題點。
- -n：告訴 make 只印出將會進行的工作，而不真正去編譯。
- -f <filename>：告訴 make 該使用的 makefile 檔案。如果不使用這個選項，make 會依序尋找目錄中的 GNUmakefile, makefile, Makefile。

Makefile的組成

- Makefile 主要內容包含了五個東西：
  - 顯式規則：顯式規則表示如何生成一個或多個目標文件。
  - 隱式規則：比較簡略地書寫 Makefile 規則，例如規則中有 .o 文件，make 會自動的把 .c 文件也加入依賴關係中。
  - 變數定義：類似 C 語言中的 define，定義的變數都會置換到引用位置上。
  - 文件指示：
    - 類似 C 語言中的 include，一個 Makefile 中引用另一個 Makefile，如 include makefile.inc。
    - 類似 C編譯器中的 #if，根據某些情況指定 Makefile 中的有效部分。
  - 註釋：只有行註釋，用 # 符號表示
  - 接續下一行符號使用 \ 符號，以\做結尾的敘述行會被視為與下一行接續在一起。

Makefile -顯式規則

- 語法：

```
target1: 相依檔案1 相依檔案2 …
<TAB>command;
target2: dependencies of target2
<TAB>command;
…
…
clean:
<TAB>rm -rf *.o
```

- 標的 (Target)：一個標的檔，可以是 Object 檔，也可以是執行檔，還可以是一個標籤。
- 相依 (Dependency, Prerequisites)：要產生標的檔 (target)相依哪些檔案。
- 命令 (Command)：建立專案時需要執行的 shell 命令。
  - 命令部分的每行的縮進必須要使用 Tab 鍵而不能使用多個空格。
- 可以使用「\」表示續行。注意，「\」之後不能有空格。

Makefile範例1

```
# Makefile1
program: main.o sub0.o sub1.o
	$(CC) -o program main.o sub0.o sub1.o
```

1. 刪除先前產生的 .o 檔及執行檔 program。
   $ rm *.o program
2. 執行 make 並指定Makefile1進行編譯。
   $ make -f Makefile1
3. 執行結果：
   $ ./program
4. 再執行一次 make，沒有任何變動，所以沒有必要進行任何重編譯。
   $ make -f Makefile1

```
os05209@os05209-VirtualBox:~/myCode/make1$ rm *.o program
os05209@os05209-VirtualBox:~/myCode/make1$ make -f Makefile1
cc     -c -o main.o main.c
cc     -c -o sub0.o sub0.c
cc     -c -o sub1.o sub1.c
cc -o program main.o sub0.o sub1.o
os05209@os05209-VirtualBox:~/myCode/make1$ ./program
main
0000
1111
os05209@os05209-VirtualBox:~/myCode/make1$ make -f Makefile1
make: 'program' is up to date.
os05209@os05209-VirtualBox:~/myCode/make1$
```

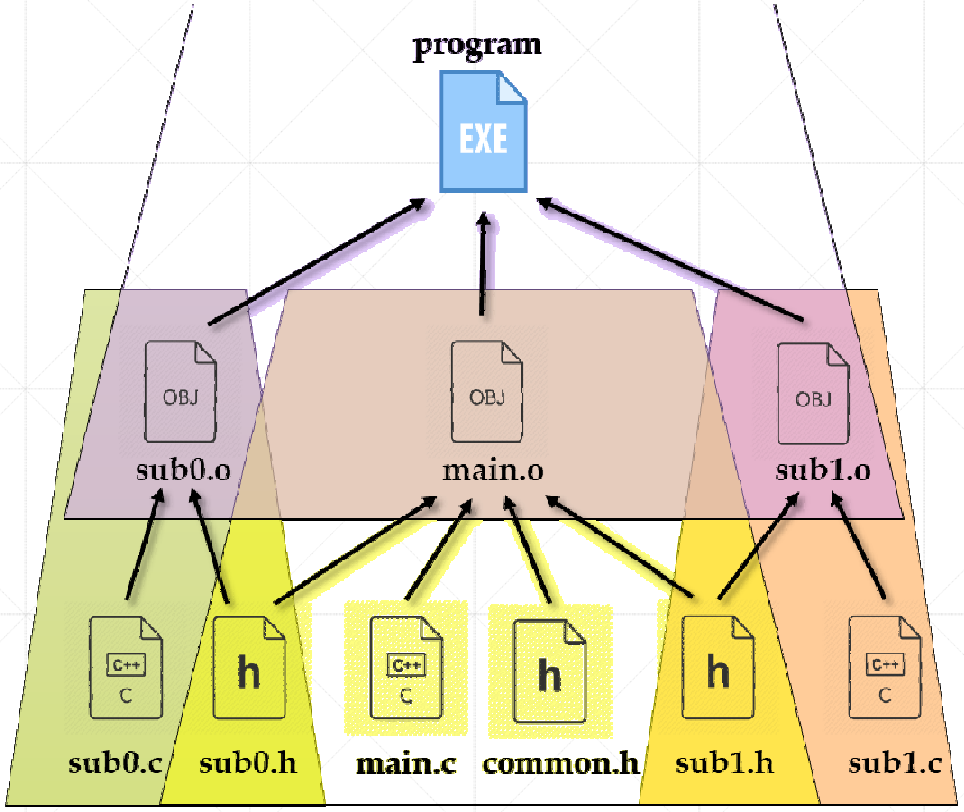
Makefile範例2-加入相依性

```
# Makefile2
program: main.o sub0.o sub1.o
	$(CC) -o program main.o sub0.o sub1.o
main.o: main.c common.h sub0.h sub1.h
sub0.o: sub0.c sub0.h
sub1.o: sub1.c sub1.h
```

1. 將編譯main.o、sub0.o及sub1.o的相依性加入Makefile1。
2. 執行 make 並指定Makefile2進行編譯。

   $ make -f Makefile2

```
os05209@os05209-VirtualBox:~/myCode/make1$ make -f Makefile2
cc     -c -o main.o main.c
cc     -c -o sub0.o sub0.c
cc     -c -o sub1.o sub1.c
cc -o program main.o sub0.o sub1.o
```



Makefile範例3-多重目標項目

- 可在 makefile 中建立多個目標項目，並於 make 時指定目標項目。

1. 增加 clean 目標項目，移除不想要的目的檔，clean 冒號之後是空白，目標項目永遠會被認為過期，所以它的命令一定會被執行。
2. 測試目標項目 clean
   $ make -f Makefile3 clean
3. 再 make 建立 program 目標檔
   $ make -f Makefile3
4. make 可先清除目標檔再編譯程式 program 目標檔
   $ make -f Makefile3 clean program

```
# Makefile3
program: main.o sub0.o sub1.o
	$(CC) -o program main.o sub0.o sub1.o
main.o: main.c common.h sub0.h sub1.h
sub0.o: sub0.c sub0.h
sub1.o: sub1.c sub1.h
clean:
	rm -f program main.o sub0.o sub1.o
```

```
os05209@os05209-VirtualBox:~/myCode/make1$ make -f Makefile3 clean
rm -f program main.o sub0.o sub1.o
os05209@os05209-VirtualBox:~/myCode/make1$ make -f Makefile3
cc     -c -o main.o main.c
cc     -c -o sub0.o sub0.c
cc     -c -o sub1.o sub1.c
cc -o program main.o sub0.o sub1.o
os05209@os05209-VirtualBox:~/myCode/make1$ make -f Makefile3 clean program
rm -f program main.o sub0.o sub1.o
cc     -c -o main.o main.c
cc     -c -o sub0.o sub0.c
cc     -c -o sub1.o sub1.c
cc -o program main.o sub0.o sub1.o
os05209@os05209-VirtualBox:~/myCode/make1$
```

Makefile範例4-內建法則

- make內建的法則是借助檔尾（suffix）得知使用的法則。
- 檔尾和符號 (pattern) 法則有下列兩項表示法，將 .old_suffix 的檔案變成 .new_suffix 的檔案。
  - .<old_suffix>.<new_suffix>:
  - %.<new_suffix>: %.<old_suffix>
- 可以利用 -p 選項，要求 make 印出內建法則，如：

```
.c.o:
        $(COMPILE.c) $(OUTPUT_OPTION) $<
%.o: %.c
        $(COMPILE.c) $(OUTPUT_OPTION) $<
```

Makefile範例4-內建法則

- 若不使用內建法則，可以於 make 時，直接加入參數，如：

  $ make CC=gcc CFLAGS="-Wall -g" -f Makefile3 program

```
os05209@os05209-VirtualBox:~/myCode/make1$ make CC=gcc CFLAGS="-Wall -g" -f Makefile3 program
gcc -Wall -g    -c -o main.o main.c
gcc -Wall -g    -c -o sub0.o sub0.c
gcc -Wall -g    -c -o sub1.o sub1.c
gcc -o program main.o sub0.o sub1.o
os05209@os05209-VirtualBox:~/myCode/make1$
```

Makefile範例5-自訂法則1

1. 自訂法則1：將 .c 檔案變成 .o 檔案。其中變數 $< 代表目前的相依性項目，會被展開成為原本的檔案名稱（含舊的檔尾）。
2. 執行 make，gcc 以 -W 選項編譯，表示法則1生效。

```
# Makefile5-1
program: main.o sub0.o sub1.o
	$(CC) -o program main.o sub0.o sub1.o
main.o: main.c common.h sub0.h sub1.h
sub0.o: sub0.c sub0.h
sub1.o: sub1.c sub1.h
clean:
	rm –f program main.o sub0.o sub1.o
.c.o:
	gcc -W -c $<
```

```
os05209@os05209-VirtualBox:~/myCode/make1$ make -f Makefile5-1
gcc -W -c main.c
gcc -W -c sub0.c
gcc -W -c sub1.c
cc -o program main.o sub0.o sub1.o
os05209@os05209-VirtualBox:~/myCode/make1$
```

Makefile範例5-自訂法則2

1. 自訂法則2：將 .c 檔案變成 .o 檔案法則改用此法則。
2. 執行 make，gcc 以 -W 選項編譯，表示法則2生效。

```
# Makefile5-2
program: main.o sub0.o sub1.o
	$(CC) -o program main.o sub0.o sub1.o
main.o: main.c common.h sub0.h sub1.h
sub0.o: sub0.c sub0.h
sub1.o: sub1.c sub1.h
clean:
	rm -f program main.o sub0.o sub1.o
%.o: %.c
	gcc -W -c $<
```

```
os05209@os05209-VirtualBox:~/myCode/make1$ make -f Makefile5-2
gcc -W -c main.c
gcc -W -c sub0.c
gcc -W -c sub1.c
cc -o program main.o sub0.o sub1.o
os05209@os05209-VirtualBox:~/myCode/make1$
```

Makefile變數使用

- 變數定義：MACRONAME=value
  - 變數與變數內容以『=』隔開，同時兩邊可以具有空格；
    (shell 中的變數設定，兩邊不可以具有空格)
  - 習慣上，變數最好是以『大寫字母』為主；
  - 等號後面的 value 變成空白時，代表將變數清成空白。
- 變數引用可以用$(MACRONAME)或是${MACRONAME}來表示皆可。
- 環境變數取用規則：
  - make 指令列後面加上的環境變數為第一優先
  - makefile 裡面指定的環境變數為第二優先
  - shell 原本具有的環境變數為第三優先

Makefile 變數範例1

1. 設定 CC、CFLAGS、OBJS 三個變數
   - CC 代表編譯器命令
   - CFLAGS代表編譯旗標選項
   - OBJS 指定所有的 .o 檔
2. make先清除目標檔再編譯程式 program 目標檔

   $ make -f Makefile6 clean program

```
# Makefile6
CC = gcc
CFLAGS = -Wall
OBJS = main.o sub0.o sub1.o

program: $(OBJS)
	$(CC) -o program $(OBJS)
main.o: main.c common.h sub0.h sub1.h
	$(CC) $(CFLAGS) -c main.c
sub0.o: sub0.c sub0.h
	$(CC) $(CFLAGS) -c sub0.c
sub1.o: sub1.c sub1.h
	$(CC) $(CFLAGS) -c sub1.c
clean:
	rm -f $(OBJS) program
```

```
os05209@os05209-VirtualBox:~/myCode/make1$ make -f Makefile6 clean program
rm -f main.o sub0.o sub1.o
gcc -Wall -c main.c
gcc -Wall -c sub0.c
gcc -Wall -c sub1.c
gcc -o program main.o sub0.o sub1.o
os05209@os05209-VirtualBox:~/myCode/make1$
```

Makefile 常用的變數及符號

符號	說明
$?	代表需要重建（被修改）的相依性項目
$@	代表目前的目標項目名稱
$<	代表目前的相依性項目
$*	代表目前的相依性項目，不過不含副檔名
-	make 會忽略命令的錯誤
@	make 不會在標準輸出 stdout 顯示要執行的命令

```
program: $(OBJS)
	$(CC) -o program $(OBJS)
```

同義

```
program: $(OBJS)
	$(CC) -o $@  $(OBJS)
```

課堂練習

- 下載makeLab.zip，為一程式專案的原始碼，請依照原始碼檔案間的相依性，參考Makefile6範例以最少修改的方式達成以下要求的結果。
  - 可編譯出執行檔為test的標的。
  - 使用目標項目 remove 來移除所產生的執行檔及所有 .o 檔。
- 將包含以下指令執行過程的畫面擷取下來。

```
$ make          <--------產生執行檔test
$ ./test        <--------執行test程式
$ make remove <--------移除所產生的執行檔及所有 .o 檔
```

- 將撰寫完成的makefile及測試畫面兩個檔案上傳至moodle。

```
# Makefile6
CC = gcc
CFLAGS = -Wall
OBJS = main.o sub0.o sub1.o

program: $(OBJS)
	$(CC) -o program $(OBJS)
main.o: main.c common.h sub0.h sub1.h
	$(CC) $(CFLAGS) -c main.c
sub0.o: sub0.c sub0.h
	$(CC) $(CFLAGS) -c sub0.c
sub1.o: sub1.c sub1.h
	$(CC) $(CFLAGS) -c sub1.c
clean:
	rm -f $(OBJS) program
```