

程式設計(二)-HW04

Due to 05/26 PM 11:59／授課老師：紀博文

一、基本資料

姓名：林育辰

系級：資工 111

學號：40771131H

二、檔案有哪些？

40771131H_hw04(主資料夾)	test(次要資料夾)
myawk.c (第一題)	Helloworld.c
test.txt (第一題)	Makefile
myjpgthumb.c (第二題)	Configure.ac
img_with_thumbnail.jpg (第二題)	其它主要是使用 Autoconf 建置組態時的各種檔案
img_hide.c (第三題)	
output.bmp(第三題測試用)	
test_out.bmp (第三題測試用)	
hw0404.c (第四題)	
Makefile	
README.pdf	

◎每個.c 檔皆有註解！請格外注意 README 中的格式要求

三、如何執行？

請輸入 make

→編譯 myawk.c, myjpgthumb.c, img_hide.c, hw0404.c

→產生 myawk, myjpgthumb, img_hide, hw0404 檔

指令如下：

\$ make

\$./myawk

\$./myjpgthumb

\$./img_hide

\$./hw0404

四、索引

第一題	P.3-6
第二題	P.7-9
第三題	P.9-16
第四題	P.17-21
第五題	P.22-29

說明

1 Simple AWK (25 pts)

AWK is a domain-specific language designed for text processing and typically used as a data extraction and reporting tool. It is a standard feature of most Unix-like operating systems. AWK was initially developed in 1977 by Alfred Aho (author of `egrep`), Peter J. Weinberger (who worked on tiny relational databases), and Brian Kernighan; it takes its name from their respective initials.

Awk is a useful tool to extract wanted data from formatted sources. I will give you a short demonstration in the class.

Now I want you to develop AWK ... of course not. I want you to develop a simple version. Given a file as `test.txt` follows.

◎題意說明→

AWK 是一個經常使用於文本處理、數據提取等功能的指令(語言)。他是大多數類 Unix 作業系統的標準功能，總而言之他最初是 1977 年開發的，相關歷史就先行跳過了。

AWK 提取有「特定格式」的數據是一個非常好用的工具，在課堂上也有示範過了，現在你應該開發一個 AWK，當然不是開發整個 AWK，只是需要你們開發一個簡單的 AWK 版本。

現在，我給一個 `test.txt` 的檔案，有以下內容：

```
Alice 70 60 50 40
Bob 100 101 102 103
Cathy 1 2 3 4
```

你的程式運作後應該要跟以下的結果相同：

```
1 ./myawk '$1' '-->' '$3' test.txt
2 Alice-->60
3 Bob-->101
4 Cathy-->2
5 ./myawk '$4' '+++' '$2' test.txt
6 50+++70
7 102+++100
8 3+++1
```

`$1` and `$3` are used to the first and the third fields which are separated by blanks. We use **quotation mark** to present the string.

You also need to support **-F** option to set **one character separator**. The default separator is blank. The option **-F** must be put after **myawk**. The last argument must be the file name.

上圖的\$1, \$3 表示第一行和第三行(藉由一個空格所分割)，此外，我們使用「單引號」來表示是一個字串。

同時，你也應該支援-F 的 option，讓使用者可以設定「分割字元」(只允許設定一個字元)，若使用者沒有使用 option，則預設為空格，option -F 會用在 myawk 的後面，最後的參數一定是一個檔案名稱，例如：

```
./myawk -F ' ' '$1' '→' '$3' test.txt
```

※注意

- 1) 課堂上說過，輸入不會刻意刁難，例如 ./myawk -F' ' 這種輸入是不需檢查的
- 2) option 的使用會緊跟著 ./myawk，其中-F 後面必定要跟著一個參數(分割字元)
- 3) 兩個\$+數字的參數，是有先後順序的，以下範例將說明
- 4) 分割字元必定恰有一個字元，不可輸入空白，或更多的字元進去
- 5) 若檔案中分割字元不存在，則整行輸出(包含換行字元)，加入使用者想輸入的字串，其後為空白，例如：

```
yuchen0515@mathlin:~/Programming/hw04$ ./myawk -F ' ' '$1' '→' '$3' test.txt
Alice 70 60 50 40
-->
Bob 100 101 102 103
-->
Cathy 1 2 3 4
-->
```

- 6) \$後的數字超出檔案的行數，該行直接不印出

◎輸出&輸出格式

編譯後，執行” \$./myawk”

其結果依據 test.txt 而定，在此使用題本所描述的內容

#1

```
yuchen0515@mathlin:~/Programming/hw04$ ./myawk '$1' '→' '$3' test.txt
Alice-->60
Bob-->101
Cathy-->2
```

以上測資，即會從 test.txt 檔案中，提取第一行 + ‘-->’ + 第三行 輸出

#2

```
yuchen0515@mathlin:~/Programming/hw04$ ./myawk '$4' '+++' '$2' test.txt
50+++70
102+++100
3+++1
```

該測資則會提取第四行 + '+++' + 第二行輸出

#3

```
yuchen0515@mathlin:~/Programming/hw04$ ./myawk -F ' ' '$1' '-->' '$3' test.txt
Alice-->60
Bob-->101
Cathy-->2
```

將第一筆測資，設定分割字元為空格，則輸出同測資 1

#4

若 test.txt 改為以下內容：

```
1 Alice .70 .60 .50 .40
2 Bob .100 .101 .102 .103
3 Cathy .1 .2 .3 .4
```

並將-F 參數設為一個點，則輸出如下：

```
yuchen0515@mathlin:~/Programming/hw04$ ./myawk -F '.' '$1' '-->' '$3' test.txt
Alice -->60
Bob -->101
Cathy -->2
```

切勿輸入不符格式的分割字元

test.txt 的每行開頭、結尾不得輸入分割字元，例如：

```
1 .Alice .70 .60 .50 .40.
2 .Bob .100 .101 .102 .103.
3 .Cathy .1 .2 .3 .4.
```

#5

(奇怪的輸入—無輸入檔名)

```
yuchen0515@mathlin:~/Programming/hw04$ ./myawk -F ' ' '$1' '-->' '$3'
Your file don't be opened!
Error: : No such file or directory
```

#6

(奇怪的輸入—過多的參數)

```
yuchen0515@mathlin:~/Programming/hw04$ ./myawk -F ' ' '$1' '-->' '$3' test.txt '$3'  
Your input error!
```

(奇怪的輸入—過少的參數)

```
yuchen0515@mathlin:~/Programming/hw04$ ./myawk -F ' '  
Your input error!
```

#7

(奇怪的輸入—格式錯誤)

```
yuchen0515@mathlin:~/Programming/hw04$ ./myawk -F ' ' '$1' '-->' '$teaching' test.txt  
Your argument should be "$" + "integer".
```

(用了 option 卻沒設定分割字元)

```
yuchen0515@mathlin:~/Programming/hw04$ ./myawk -F '$1' '-->' '$3' test.txt  
You can only to set "one" character
```

(分割字元只能輸入一個字元)

```
yuchen0515@mathlin:~/Programming/hw04$ ./myawk -F 'test' '$1' '-->' '$3' test.txt  
You can only to set "one" character
```

◎程式設計思路

- 1) 檢查使用者起初輸入有沒有「-」的符號，若有就用 getopt 檢查參數
- 2) 各種檢查使用者輸入完畢後開始讀檔，使用 strtok 分割並輸出即可。

說明

2 JPEG File Interchange Format Thumbnail (25 pts)

The JPEG format is as follows. Do not worry, I will not ask you to decode it. Note that a JPEG can embed a thumbnail. Please write a program to embed (overwrite) a BMP thumbnail into a JPG and extract a thumbnail from a JPG into a BMP.

<https://zh.wikipedia.org/zh-tw/JPEG%E6%96%87%E4%BB%B6%E4%BA%A4%E6%8D%A2%E6%A0%BC%E5%BC%8F>

◎題意說明

→JPEG 的格式如下面給的 wikipedia 的內容。噢！不用擔心，我沒有要你解碼 JPEG，你可以注意到 JPEG 可以嵌入縮圖。請寫出一個程式能夠嵌入一個 BMP 檔變成 JPEG 內的縮圖，也可以從 JPG 中提取縮圖變為一個 BMP 檔

```
1 ./myjpgthumb [option] [jpg] [bmp]
2 -w --write: (Over)Write the bmp file to the jpg file's thumbnail.
3 -e --extract: Extract the jpg file's thumbnail to the bmp.
```

→此外，這個程式應該要能夠讓使用者輸入參數去執行，option 可以輸入 -w, -e 兩種指令，其中也可用 -write 表示 -w，--extract 表示 -e，其後必須輸入兩個檔案，依序是 jpg 的檔名、bmp 的檔名

注意：

- (1) jpg, bmp 檔名一定要輸入，若使用 write，則必定要有一個 bmp 檔案，反之，若使用 extract，則 jpg 內應包含縮圖資訊，提取後會產生一個 bmp 檔案，其檔名為使用者所輸入的名稱
- (2) 當使用 write 時，請注意，jpg 中的縮圖資訊只可放最大高 255, 寬 255 的圖片，特別的是，縮圖的「高」應為 4 的倍數，否則開啟 bmp 時容易出現問題，若您想使用 write 去存入一個 bmp 檔進入 jpg 的縮圖區域，bmp 檔高、寬都不得超出 255，否則不予處理
- (3) 進行處理時，本程式會將 JPG 檔備份為「backup.jpg」的檔案，以避免錯誤造成原檔損毀
- (4) 只允許 --extract, --write, -e, -w 的 option，前面的 - 不得省略
- (5) 請注意，輸入的 bmp 檔案，高度不得為「負數」，不處理這個情況(一般來說負數會讓圖片檢視器以「上下」相反的方式解讀)

◎輸入&輸出格式

編譯後，執行” `$./myjpgthumb`”，

其後應輸入 option (-w or --write, -e --extract)和兩個檔名(jpg, bmp)

在此使用老師提供的「img_with_thumbnail.jpg」

```
yuchen0515@mathlin:~/Programming/hw04$ ./myjpgthumb --extract img_with_thumbnail.jpg output.bmp
Extract the jpg file's thumbnail to the bmp.....
Done!
```

p.s. 在此有稍微製作類似進度條的效果，出現 Done 表示成功

提取後檔案如圖：



(很多人顏色會變，主要是因為 jpg 和 bmp 儲存像素的順序不同，一個為 bgr，一個為 rgb，請格外注意，一般讀出後 bmp 是相反的圖片，因此這邊將標頭檔的 height 調為負數，讓他自動翻轉)

(寫入)

```
yuchen0515@mathlin:~/Programming/hw04$ ./myjpgthumb --write img_with_thumbnail.jpg output.bmp
(Over)Write the bmp file to the jpg file's thumbnail.....
Done!
```

請注意，寫入當前目錄必須要有該 bmp 檔，且大小 $x \leq 255$, $y \leq 255$

若 bmp 檔案輸入錯誤則會顯示如下：

```
yuchen0515@mathlin:~/Programming/hw04$ ./myjpgthumb --write img_with_thumbnail.jpg output.bmp
(Over)Write the bmp file to the jpg file's thumbnail.....
File could not be opened!
```


◎程式設計思路

- 1) 用 `getopt_long` 去處理參數問題，即可順利讀取 `-w -write, -e -extract`
 - 2) 一般的檢查完畢後，若使用者輸入 `-extract`，則在 `jpg` 檔中找識別碼「4A 46 49 46 00」的區段(JPIF)，因為若圖檔有 `JPG` 擴充碼則會在 `JPIF` 前面，會導致讀取出現問題，因此利用識別碼找到正確的位置，其後 7 的 bytes 後會存縮圖的 `x, y` 長度，接著會出現 `x*y` 的內容(圖檔內容)
 - 3) 寫好標頭檔，存進去即可，若為 `write` 則需讀取掉 `bmp` 的標頭檔再寫入，特別注意 `jpg` 縮圖資訊後方的資料不可遺漏
-

說明

3 Image Steganography (25 pts)

Steganography is the practice of concealing a file, message, image, or video within another file, message, image, or video. In this problem, I want you to implement a simple image data hiding program.

The concept is simple. Given a `bmp` image file, where `R, G, B` are presented in 8 bits. To embed a secret data in the cover `bmp` file, you just replace the last significant bits with the secret data bit, as shown in figure 1. If you want more capacity, you can use the last two significant bits, as shown in figure 2. Since we do not want to affect the original image file too much, the embedding order is from `LSB` to `MSB`.

Now I want you to embed and extract a secret data in/from the cover `bmp` image file.

◎題意說明

→ 圖片隱寫術能夠將一個檔案、訊息、圖檔、或影片藏在另一個檔案內，因此，我想要你們去實現一個簡單的程式，能夠將你的程式檔藏進一個圖片中。

概念很簡單，給定一個 `bmp` 的圖檔，其中 `RGB` 各佔了 8bits，如果你想藏東西在裡面，你只需要替換掉 8bits 中的最後幾個 bits(影響力儘可能的小)，可以只換掉 1bits，如果想要更大的容納空間，你也可以換掉 2bits, 3bits...

因為我們不想要影響原始圖檔太多，因此藏的順序從最小位元到最大位元。

現在，我想要你們能夠從一個 `bmp` 的檔案藏／提取祕密檔案。

```

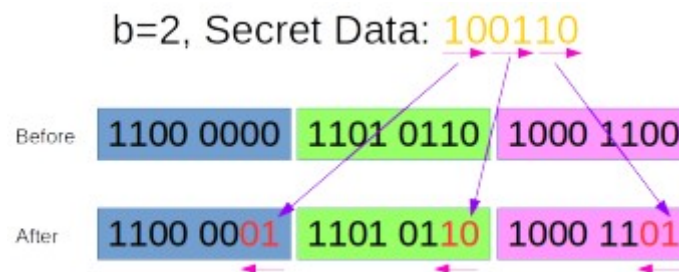
1 ./img_hide [option] [cover_bmp] [secret_data]
2 -w, --write: Write the secret data to the cover_bmp.
3 -e, --extract: Extract the secret data from the cover_bmp to the secret_data.
4 -b, --bits=N: use last N bits. N is from 1 to 8. The default N is 1.

```

使用者可以輸入 option，-w, --write 都表示寫入，-e, --extract 都代表提取，-b, --bits 代表選擇藏的 bits 數($1 \leq x \leq 8$)

1. The secret data size should be recorded in the bmp header reserved field which is 4 bytes long.

祕密檔案的大小(檔案大小)應該紀錄在 bmp 的標頭檔內(reserve 那一欄)，長度為 4 bytes。



若 b 輸入為 2，則會將祕密資料塞在 R,G,B 的最後兩個位元，每次祕密資料以 1byte 為單位，如為上述的資料 100110，則存入的是 00100110(否則難以切割資料)，預設為 1 個位元

注意：

- (1) option 中，只可輸入 **write, extract** 其中一個選項，-b 的使用如下：

```
$ ./img_hide -w -b 3 [cover_bmp] [secret_data]
```

-b 後面代表選取的位元數，若使用-b 後面必定要給位元參數

(老師上課表示，這裡使用者輸入都會符合格式，不會刁難)

- (2) 若 cover_bmp 不夠藏 secret_data，會印出錯誤訊息
- (3) 資料夾內，若需寫入，則 **secret_data** 和 **cover_bmp** 皆需存在，若想提取，則 cover_bmp 必須存在，secret_data 可以自己命名
- (4) bits 若不在[1,8]的範圍內，則提示錯誤並退出程式

◎輸入&輸出格式

編譯後，執行” \$./img_hide”

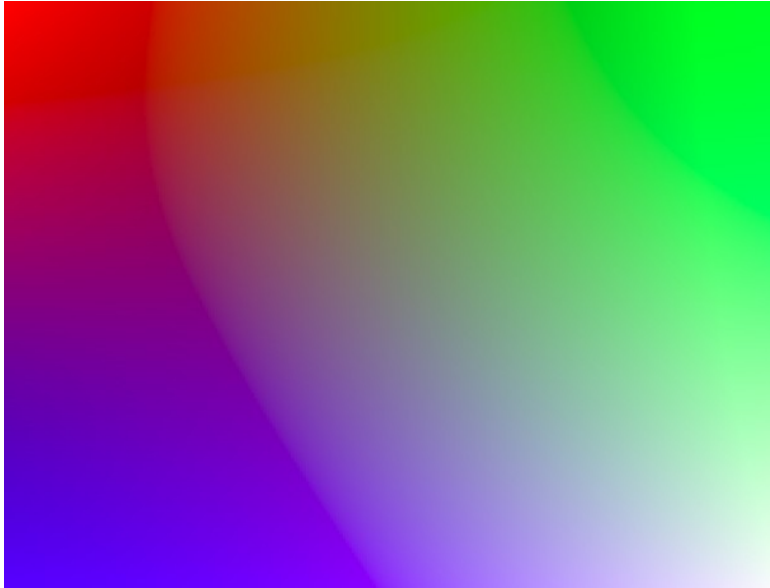
其後應輸入 option (-w or --write, -e --extract)

若需要可以輸入位元數(-b, --bits + 數字[1-8])

以及兩個檔名[cover_bmp] [secret_data]

在此使用作業三產生的圖檔，以及本次作業第二題提取的縮圖檔，作為測試：

[cover_bmp] (output.bmp)



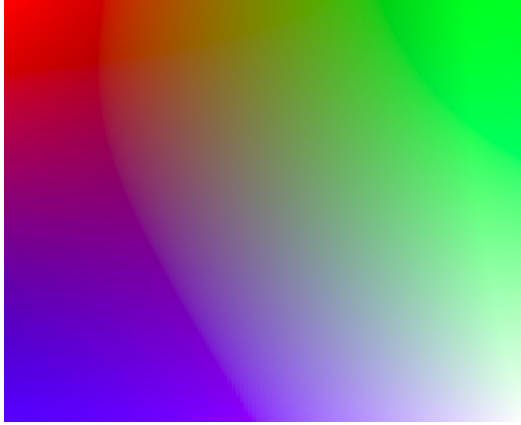
[secret_data] (test_out.bmp)



◎測試一：存入 1 bit

```
yuchen0515@mathlin:~/Programming/hw04$ ./img_hide -w output.bmp test_out.bmp  
Done!
```

將 test_out.bmp 的檔案，藏進 output.bmp 的後面 1 bit 內



(因為只有 1bit，基本上看不太出來有什麼差異)

提取

```
yuchen0515@mathlin:~/Programming/hw04$ ./img_hide -e output.bmp test_out.bmp  
Done!
```

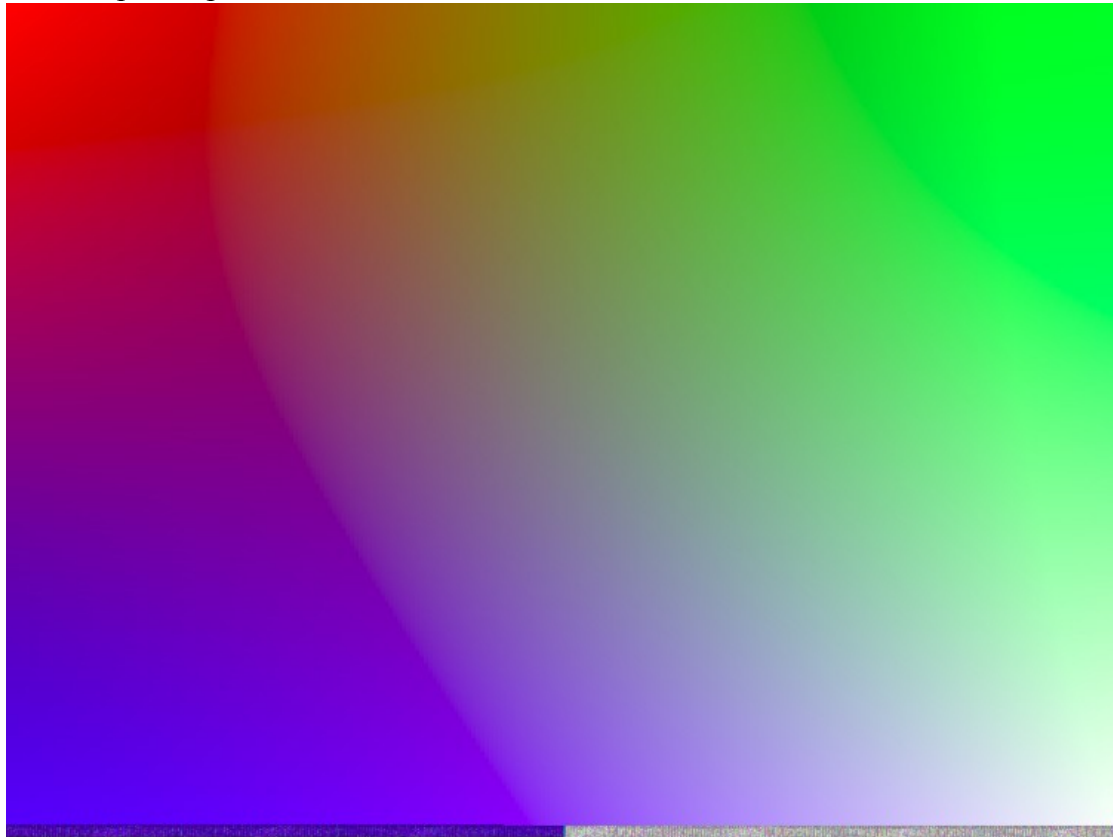
從藏的資料裡得到原始圖檔



◎測試二：存入 7 bits

```
yuchen0515@mathlin:~/Programming/hw04$ ./img_hide -w -b 7 output.bmp test_out.bmp  
Done!
```

存進 output.bmp 檔案的後 7bits



因為塞了 7bits，雖主要色調不變，但很多細微的顏色就被吃掉了

```
yuchen0515@mathlin:~/Programming/hw04$ ./img_hide -e -b 7 output.bmp test_out.bmp  
Done!
```

提取後：



◎測試三：存入程式檔，5bits

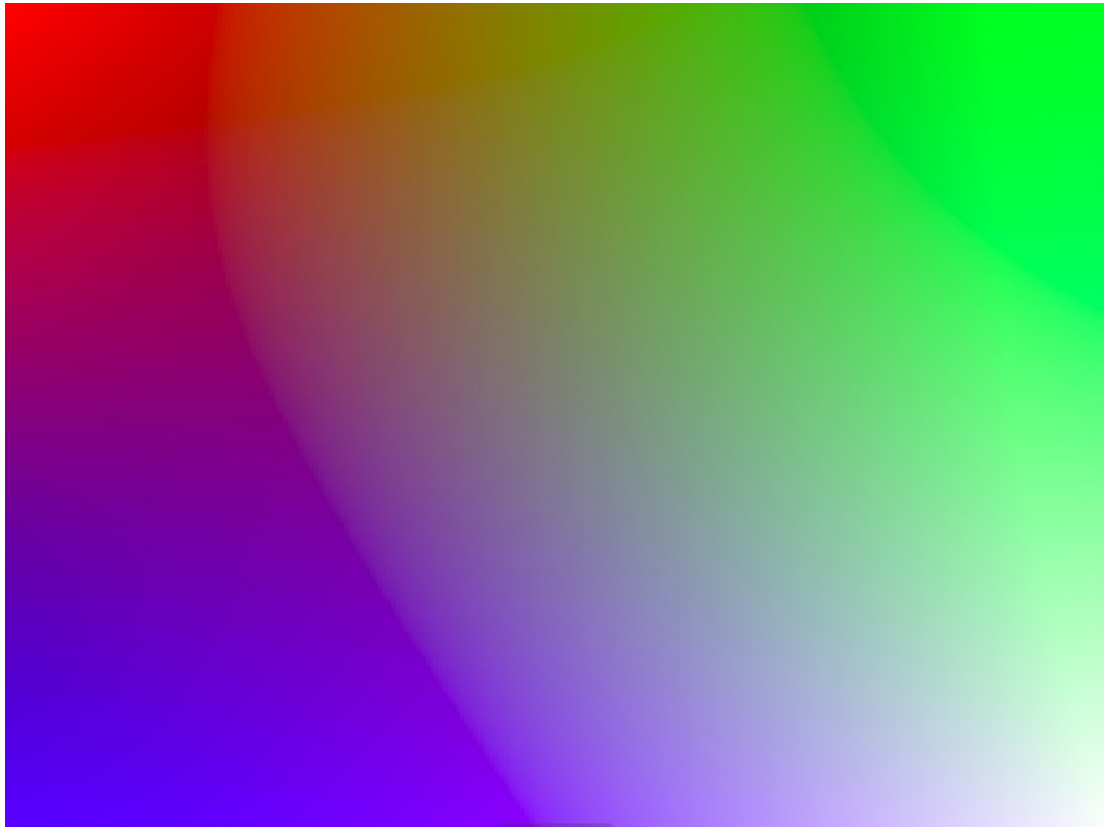
[secret_data] test3.c

```
1 #include <stdio.h>
2 #include <stdint.h>
3
4 int main(){
5     int32_t mask = 1 << 1, mask2;
6     int32_t save= 216;
7     int32_t tp = 3;
8
9     for (int32_t i = 0 ; i < 2 ; i++){
10         mask2 = (tp & mask);
11         mask >>= 1;
12         save |= mask2;
13         printf("%d\n", mask2);
14     }
15     puts("");
16     printf("%d\n", save);
17
18
19     return 0;
20 }
```

~
"test3.c" 20L, 288C

將程式檔存進去圖檔

```
yuchen0515@mathlin:~/Programming/hw04$ ./img_hide -w -b 5 output.bmp test3.c
Done!
```



主要因為程式檔較小，而且直接存入 5bits，雖 5bits 影響到圖檔蠻大的，不過因為檔案小，只有影響到左下方一點點像素，藍色有點變紫

提取：

```
yuchen0515@mathlin:~/Programming/hw04$ ./img_hide -e -b 5 output.bmp test3.c  
Done!
```

```
test3.c  
~/Programming/hw04  
1 #include <stdio.h>  
2 #include <stdint.h>  
3  
4 int main(){  
5     int32_t mask = 1 << 1, mask2;  
6     int32_t save= 216;  
7     int32_t tp = 3;  
8  
9     for (int32_t i = 0 ; i < 2 ; i++){  
10         mask2 = (tp & mask);  
11         mask >>= 1;  
12         save |= mask2;  
13         printf("%d\n", mask2);  
14     }  
15     puts("");  
16     printf("%d\n", save);  
17  
18  
19     return 0;  
20 }
```

◎程式設計思路

- 1) 先用 `optget_long` 去處理參數，以及各式檢查輸入
- 2) 為了避免圖檔走鐘太多，因此將原本 `bmp` 檔備份存成 `cover_bmp_backup` 的檔案，方便使用者重新使用
- 3) 每次都以 1 byte 為單位去讀 `secret_data`、`cover_bmp`，利用 `mask` 遮罩去拿到相對應的 `secret_data` 的位置，再用 `mask2` 弄到 `bgr` 相對應位置用 OR 運算即可
- 4) 存取也是使用相似的方式，即可完成！

關鍵在於：

- 1) **位元運算**、善用 **mask 遮罩**
- 2) 單位為 **1byte**

兩個關鍵掌握後，就會很好處理！

說明

4 MACRO (25 pts)

Please define the following macros.

- **FOR_EACH**: to replace a for loop statement. The step is 1.
- **FOR_EACH_STEP**: to replace a for loop statement. The step is given by the user.
- **SUM_OF_ARRAY**: summation of all array elements.

All macros should support nested usage.

You should also provide an example code for these three macros.

◎題意說明

→ 請定義以下幾個巨集：

- (1) **FOR_EACH**→可取代一個 for loop 如 (for int32_t i= 0 ; i< n ; i++)，每次 i+=1
- (2) **FOR_EACH_STEP**→同上，其中 i 每次增加的數，由使用者決定(必定>0)
- (3) **SUM_OF_ARRAY**→加總陣列中所有元素

以上的巨集必須支持巢狀的使用，你應該為這三個巨集提供一個範例程式

※使用方法

→本程式撰寫的 3 個巨集寫法與用法

一、FOR_EACH(var, i, j) [範例一]

輸入

>>var→迴圈內變數名稱(如平常所用的 i, j, k)

>>i→起始值

>>j→結束值

```
//單層迴圈, STEP == 1
printf("-----單層迴圈, i = 0 ~ 10, i++ -----\\n");
FOR_EACH(i, 0, 10){
    printf("%d ", i);
}
puts("\\n\\n");
```

此處演示如何使用 FOR_EACH 撰寫單層迴圈，

其中後方三個參數，分別為「變數名稱」、「起始值」、「終止值」

結果如圖：

```
yuchen0515@mathlin:~/Programming/hw04$ ./hw0404
-----單層迴圈, i = 0 ~ 10, i++ -----
0 1 2 3 4 5 6 7 8 9
```

事實上他代表著這樣的 code (我們使用 gcc -E 將他展開)

```
printf("-----單層迴圈, i = 0 ~ 10, i++ -----\\n");
for(int32_t i = 0 ; i < 10; i++){
    printf("%d ", i);
}
puts("\\n\\n");
```

使用時要注意，此巨集只有取代 for (int i = 0 ; i < n ; i++) 這一段程式

至於大括號，內容還是要自己加入(不過省了很多文字)

(老師上課也說大括號應該是使用者自己要加的)

[範例二]

```
//巢狀迴圈, STEP == 1
printf("-----巢狀迴圈, 九九乘法表_example -----\\n");
FOR_EACH(i, 1, 11){
    FOR_EACH(j, 1, 11){
        printf("%d * %d = %d\\t", i, j, i*j);
        if (j == 5) puts("");
    }
    puts("");
}
```

此範例演示了如何使用此巨集撰寫巢狀迴圈，去實作一個九九乘法表

將此展開後，如下(gcc -E hw0404.c)

```
printf("-----巢狀迴圈, 九九乘法表_example -----\\n");
for(int32_t i = 1 ; i < 11; i++){
    for(int32_t j = 1 ; j < 11; j++){
        printf("%d * %d = %d\\t", i, j, i*j);
        if (j == 5) puts("");
    }
    puts("");
}
```

結果則如圖：

```
-----巢狀迴圈, 九九乘法表_example -----
1 * 1 = 1      1 * 2 = 2      1 * 3 = 3      1 * 4 = 4      1 * 5 = 5
1 * 6 = 6      1 * 7 = 7      1 * 8 = 8      1 * 9 = 9      1 * 10 = 10
2 * 1 = 2      2 * 2 = 4      2 * 3 = 6      2 * 4 = 8      2 * 5 = 10
2 * 6 = 12     2 * 7 = 14     2 * 8 = 16     2 * 9 = 18     2 * 10 = 20
3 * 1 = 3      3 * 2 = 6      3 * 3 = 9      3 * 4 = 12     3 * 5 = 15
3 * 6 = 18     3 * 7 = 21     3 * 8 = 24     3 * 9 = 27     3 * 10 = 30
4 * 1 = 4      4 * 2 = 8      4 * 3 = 12     4 * 4 = 16     4 * 5 = 20
4 * 6 = 24     4 * 7 = 28     4 * 8 = 32     4 * 9 = 36     4 * 10 = 40
5 * 1 = 5      5 * 2 = 10     5 * 3 = 15     5 * 4 = 20     5 * 5 = 25
5 * 6 = 30     5 * 7 = 35     5 * 8 = 40     5 * 9 = 45     5 * 10 = 50
6 * 1 = 6      6 * 2 = 12     6 * 3 = 18     6 * 4 = 24     6 * 5 = 30
6 * 6 = 36     6 * 7 = 42     6 * 8 = 48     6 * 9 = 54     6 * 10 = 60
7 * 1 = 7      7 * 2 = 14     7 * 3 = 21     7 * 4 = 28     7 * 5 = 35
7 * 6 = 42     7 * 7 = 49     7 * 8 = 56     7 * 9 = 63     7 * 10 = 70
8 * 1 = 8      8 * 2 = 16     8 * 3 = 24     8 * 4 = 32     8 * 5 = 40
8 * 6 = 48     8 * 7 = 56     8 * 8 = 64     8 * 9 = 72     8 * 10 = 80
9 * 1 = 9      9 * 2 = 18     9 * 3 = 27     9 * 4 = 36     9 * 5 = 45
9 * 6 = 54     9 * 7 = 63     9 * 8 = 72     9 * 9 = 81     9 * 10 = 90
10 * 1 = 10    10 * 2 = 20    10 * 3 = 30    10 * 4 = 40    10 * 5 = 50
10 * 6 = 60    10 * 7 = 70    10 * 8 = 80    10 * 9 = 90    10 * 10 = 100
```

二、FOR_EACH_STEP(var, i, j, step) [範例三]

輸入

>>var→迴圈內變數名稱(如平常所用的 i, j, k)

>>i→起始值

>>j→結束值

>>step→每次進行迴圈加的數

```
//單層迴圈，STEP使用者自訂
printf("-----單層迴圈, i = 0 ~ 10, i+=2 -----\\n");
int32_t step = 2;
FOR_EACH_STEP(i, 0, 10, 2){
    printf("%d ", i);
}
puts("\\n\\n");
```

上圖由使用者自行輸入 step，i 從 0 跑到 10，每次增加 2

展開後如圖：

```
printf("-----單層迴圈, i = 0 ~ 10, i+=2 -----\\n");
int32_t step = 2;
for(int32_t i = 0 ; i < 10; i+=2){
    printf("%d ", i);
}
puts("\\n\\n");
```

結果如圖：

```
-----單層迴圈, i = 0 ~ 10, i+=2 -----
0 2 4 6 8
```

三、SUM_OF_ARRAY(array, i, j, sum) [範例四]

輸入：

>>array→陣列名稱

>>i→起始值

>>j→結束值

>>sum→欲加入的變數名 (會自行宣告)

```
//陣列加總
printf("-----陣列加總, sum = x[0] + ... + x[10] -----\\n");
SUM_OF_ARRAY(x, 0, 10, sum);
printf("%d\\n", sum);
puts("\\n");
```

上圖中 x 的定義如下：

```
int32_t x[] = {1,2,3,4,5,6,7,8,9,10};
```

因此，在巨集內輸入陣列名稱，想從哪個位置加到哪個位置

並輸入要存到哪個變數即可(會自行宣告)

若要加總陣列所有數值，僅需從 0 到 n 即可

此段程式碼展開後如下：

```
printf("-----陣列加總, sum = x[0] + ... + x[10] -----\\n");
int32_t sum = 0; for(int32_t p = 0 ; p < 10 ; p++){ sum += x[p];};
printf("%d\\n", sum);
puts("\\n");
```

結果：

```
-----陣列加總, sum = x[0] + ... + x[10] -----
55
```

以上巨集的使用方式應很好理解

若要觀看 example_code 請看 hw0404.c 的檔案

說明

5 Bonus: Autoconf (5 pts)

GNU Autoconf is a tool for producing configure scripts for building, installing and packaging software on computer systems where a Bourne shell is available.

Autoconf is agnostic about the programming languages used, but it is often used for projects using C, C++, Fortran, Fortran 77, Erlang or Objective-C.

A configure script configures a software package for installation on a particular target system. After running a series of tests on the target system, the configure script generates header files and a makefile from templates, thus customizing the software package for the target system. Together with Automake and Libtool, Autoconf forms the GNU Build System, which comprises several other tools, notably Autoheader.

Please write a small tutorial to show how it works.

◎題意說明

Gnu Autoconf 是在 Bourne 的 Shell 下提供編譯、安裝和打包軟體的「組態指令碼」的一項工具。他不受程式語言的限制，用於各種常見的程式語言如 C, C++ 等。組態指令碼控制了軟體包在特定系統上的安裝，經過一連串測試後，會產生一個 configure 腳本(他會產生 configure.scan 的檔案)，以及從模板中生成 Makefile(通常是產生 Makefile.in)和標頭檔進而調整軟體包，使其能適應在某一種系統上，autoconf, automake, libtool 等軟體組成了「GNU 建構系統」

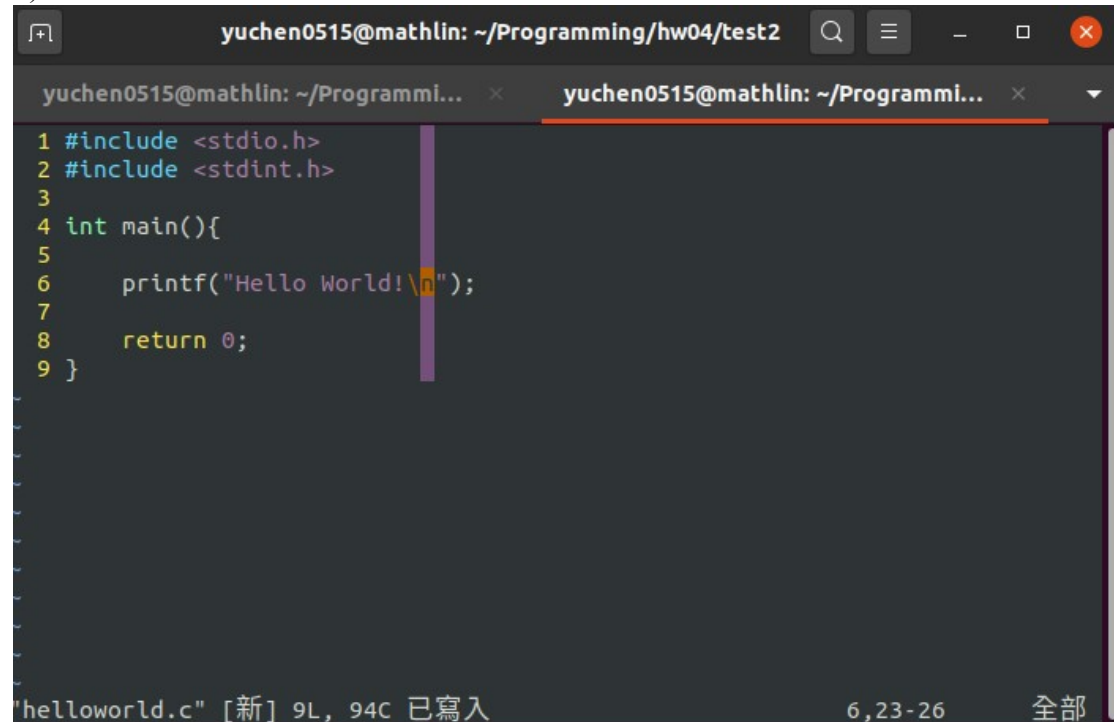
今天，要你編寫一個簡單的教學，展示他如何使用！

◎目標

→簡單撰寫一個 helloworld 的程式，並產生其 Makefile 檔案

◎步驟

1) 先用熟悉的環境，寫出一個 helloworld.c 檔，如下：



```
yuchen0515@mathlin: ~/Programming/hw04/test2
1 #include <stdio.h>
2 #include <stdint.h>
3
4 int main(){
5
6     printf("Hello World!\n");
7
8     return 0;
9 }
```

'helloworld.c' [新] 9L, 94C 已寫入 6,23-26 全部

2) 輸入 autoscan 的指令

```
yuchen0515@mathlin:~/Programming/hw04/test2$ autoscan
yuchen0515@mathlin:~/Programming/hw04/test2$ ls
autoscan.log  configure.scan  helloworld.c
yuchen0515@mathlin:~/Programming/hw04/test2$
```

應會有以上 3 個檔案，並將 configure.scan 檔名改為 configure.ac

```
autoscan.log  configure.ac  helloworld.c
```

3) 開啟 configure.ac 檔案，如下圖：

```
1 #                                     -*- Autoconf -*-
2 # Process this file with autoconf to produce a configure script.
3
4 AC_PREREQ([2.69])
5 AC_INIT([FULL-PACKAGE-NAME], [VERSION], [BUG-REPORT-ADDRESS])
6 AC_CONFIG_SRCDIR([helloworld.c])
7 AC_CONFIG_HEADERS([config.h])
8
9 # Checks for programs.
10 AC_PROG_CC
11
12 # Checks for libraries.
13
14 # Checks for header files.
15 AC_CHECK_HEADERS([stdint.h])
16
17 # Checks for typedefs, structures, and compiler characteristics.
18
19 # Checks for library functions.
20
21 AC_OUTPUT
```

將其中的第 4 行 mark 掉

第 5 行改為 `AC_INIT(helloworld.c)` (即你的檔名)

注意，請新增一行指令叫作 `AM_INIT_AUTOMAKE(helloworld, 1.0)`

前者為執行檔檔名，後者為版本編號(任意輸入即可)

第 6, 7, 15 行 mark 掉

第 21 行改為 `AC_OUTPUT(Makefile)` (即產生一個 Makefile 檔案)

更改後如圖：

```
1 #                                     -*- Autoconf -*-
2 # Process this file with autoconf to produce a configure script.
3
4 #AC_PREREQ([2.69])
5 #AC_INIT([FULL-PACKAGE-NAME], [VERSION], [BUG-REPORT-ADDRESS])
6 AC_INIT(helloworld.c)
7 AM_INIT_AUTOMAKE(helloworld, 1.0)
8 #AC_CONFIG_SRCDIR([helloworld.c])
9 #AC_CONFIG_HEADERS([config.h])
10
11 # Checks for programs.
12 AC_PROG_CC
13
14 # Checks for libraries.
15
16 # Checks for header files.
17 #AC_CHECK_HEADERS([stdint.h])
18
19 # Checks for typedefs, structures, and compiler characteristics.
20
21 # Checks for library functions.
22
23 AC_OUTPUT(Makefile)
```

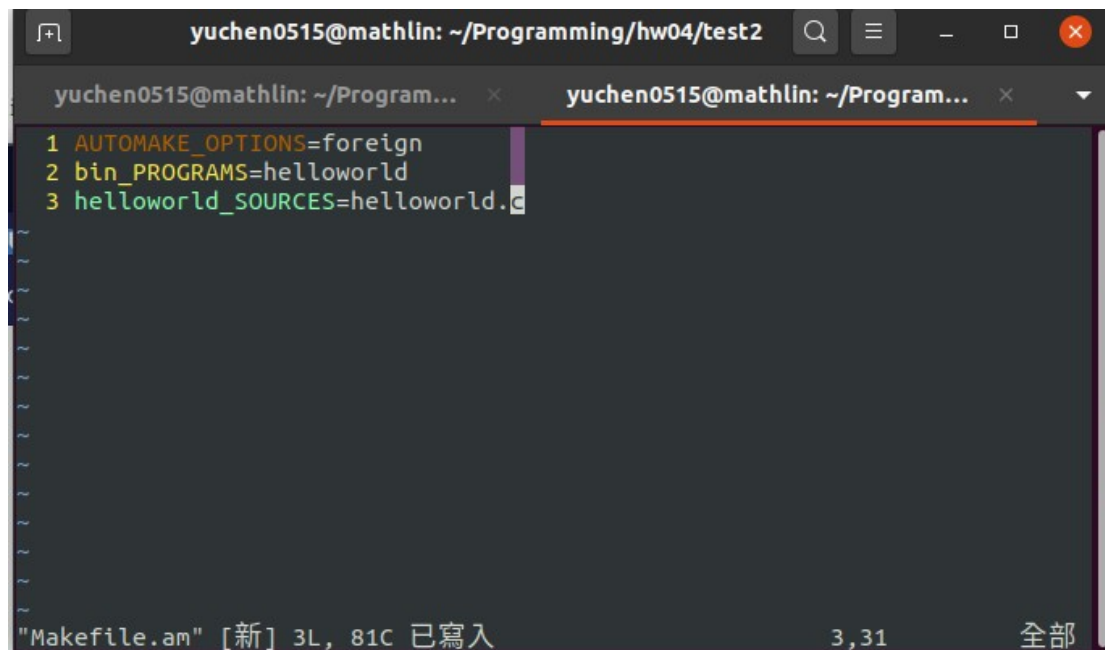
4) 輸入 aclocal 指令，應如下圖：

```
yuchen0515@mathlin:~/Programming/hw04/test2$ aclocal
yuchen0515@mathlin:~/Programming/hw04/test2$ ls
autom4te.cache  autoscan.log  configure.ac  helloworld.c
yuchen0515@mathlin:~/Programming/hw04/test2$
```

5) 輸入 autoconf 指令，如圖：

```
yuchen0515@mathlin:~/Programming/hw04/test2$ autoconf
yuchen0515@mathlin:~/Programming/hw04/test2$ ls
aclocal.m4      autoscan.log  configure.ac
autom4te.cache  configure     helloworld.c
```

6) 製作 Makefile.am 的檔案，內容則如圖：



The screenshot shows a text editor window with the title "yuchen0515@mathlin: ~/Programming/hw04/test2". The editor contains three lines of text in a Makefile format:

```
1 AUTOMAKE_OPTIONS=foreign
2 bin_PROGRAMS=helloworld
3 helloworld_SOURCES=helloworld.c
```

The status bar at the bottom indicates "Makefile.am" [新] 3L, 81C 已寫入, 3,31 全部.

第2行後面輸入可執行檔檔名，第3行後方則輸入使用到的檔案

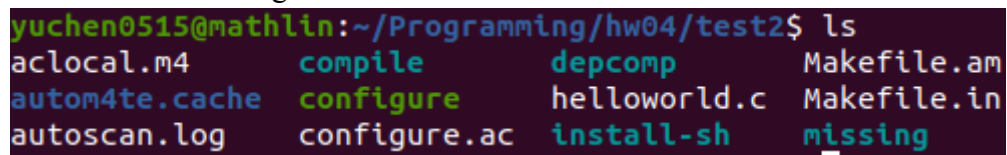
7) 輸入指令：

\$./compile

\$./install-sh

\$./missing

\$ automake --add-missing



The screenshot shows a terminal window with the command prompt "yuchen0515@mathlin:~/Programming/hw04/test2\$". The command "ls" has been executed, and the output is displayed in a grid-like format:

aclocal.m4	compile	depcomp	Makefile.am
autom4te.cache	configure	helloworld.c	Makefile.in
autoscan.log	configure.ac	install-sh	missing

其中 Makefile.in 是最重要的一個檔案(生成 Makefile 必須)

8) 輸入指令 \$./configure

```
yuchen0515@mathlin:~/Programming/hw04/test2$ ./configure
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /usr/bin/mkdir -p
checking for gawk... gawk
checking whether make sets $(MAKE)... yes
checking whether make supports nested variables... yes
checking for gcc... gcc
checking whether the C compiler works... yes
checking for C compiler default output file name... a.out
checking for suffix of executables...
checking whether we are cross compiling... no
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ISO C89... none needed
checking whether gcc understands -c and -o together... yes
checking whether make supports the include directive... yes (GNU style)
checking dependency style of gcc... gcc3
checking that generated files are newer than configure... done
configure: creating ./config.status
config.status: creating Makefile
config.status: executing depfiles commands
```

```
yuchen0515@mathlin:~/Programming/hw04/test2$ ls
aclocal.m4      compile        configure      helloworld.c  Makefile.am
autom4te.cache  config.log     configure.ac   install-sh     Makefile.in
autoscan.log    config.status  depcomp       Makefile       missing
```

你可以看到 Makefile 順利產生了！

他會編成約 739 行的 Makefile，開頭大略如下：

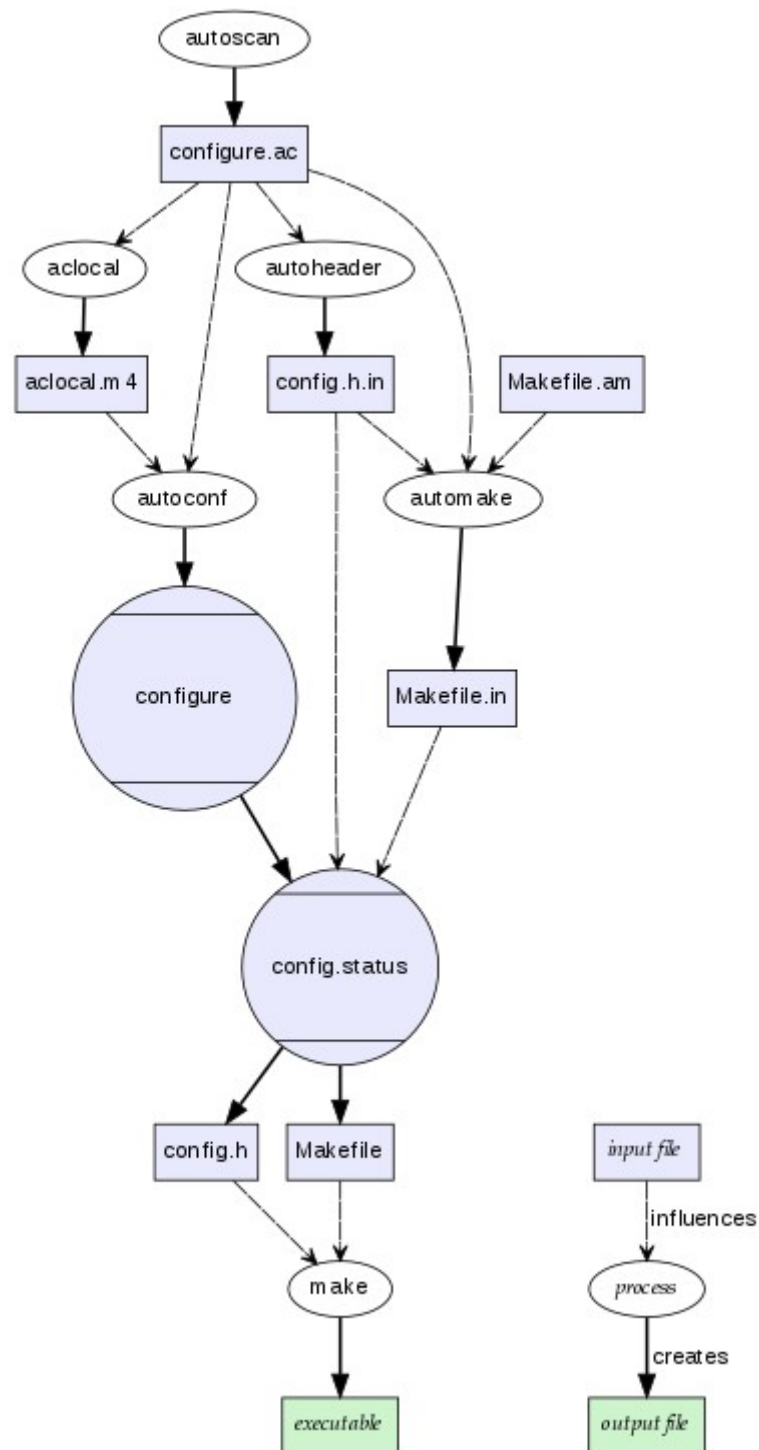
```
1 # Makefile.in generated by automake 1.16.1 from Makefile.am.
2 # Makefile. Generated from Makefile.in by configure.
3
4 # Copyright (C) 1994-2018 Free Software Foundation, Inc.
5
6 # This Makefile.in is free software; the Free Software Foundation
7 # gives unlimited permission to copy and/or distribute it,
8 # with or without modifications, as long as this notice is preserved.
9
10 # This program is distributed in the hope that it will be useful,
11 # but WITHOUT ANY WARRANTY, to the extent permitted by law; without
12 # even the implied warranty of MERCHANTABILITY or FITNESS FOR A
13 # PARTICULAR PURPOSE.
14
15
16
17
18 am__is_gnu_make = { \
19   if test -z '$(MAKELEVEL)'; then \
20     false; \
21   elif test -n '$(MAKE_HOST)'; then \
22     true; \
23   elif test -n '$(MAKE_VERSION)' && test -n '$(CURDIR)'; then \
24     true; \
25   else \
26     false; \
27   fi; \
28 }
29 am__make_running_with_option = \
30   case ${target_option-} in \
31     ?) ;; \
32     *) echo "am__make_running_with_option: internal error: invalid" \
33        "target option '${target_option-}' specified" >&2; \
34     exit 1;; \
35   esac; \
```

9) 輸入指令 \$ make, \$./helloworld

會發現這個 Makefile 檔可以順利運行、編譯，如圖：

```
yuchen0515@nathlin:~/Programming/hw04/test2$ make
gcc -DPACKAGE_NAME=\"\" -DPACKAGE_TARNAME=\"\" -DPACKAGE_VERSION=\"\" -DPACKAGE_STRING=\"\" -DPACKAGE_BUGREPORT=\"\" -DPACKAGE_URL=\"\" -DPACKAGE=\"helloworld\" -DVERSION=\"1.0\" -I. -g -O2 -MT helloworld.o -MD -MP -MF .deps/helloworld.Tpo -c -o helloworld.o helloworld.c
mv -f .deps/helloworld.Tpo .deps/helloworld.Po
gcc -g -O2 -o helloworld helloworld.o
yuchen0515@nathlin:~/Programming/hw04/test2$ ./helloworld
Hello World!
```

這是維基百科上，關於 autoconf 的流程圖：



和上面我們所執行的步驟很相近，這是一套簡單的工具能夠自動替我們產生 Makefile，根據系統、軟體包等等去調適，使其能夠適應在我們的系統上，成功運行 make。