HW7

409510095　黃偉城

甲. 說明你的執行環境（請自己想一下，哪些會和結果的正確語法有關，列出有關的執行環境就好）。

```
eric@eric-VivoBook-ASUSLaptop-X512FL-S512FL:~/Desktop/OS/HW7$ vim find_ver.c
eric@eric-VivoBook-ASUSLaptop-X512FL-S512FL:~/Desktop/OS/HW7$ gcc find_ver.c
eric@eric-VivoBook-ASUSLaptop-X512FL-S512FL:~/Desktop/OS/HW7$ ./a.out
201710
eric@eric-VivoBook-ASUSLaptop-X512FL-S512FL:~/Desktop/OS/HW7$ gcc --version
gcc (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0
Copyright (C) 2019 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

eric@eric-VivoBook-ASUSLaptop-X512FL-S512FL:~/Desktop/OS/HW7$
```

我的執行環境是 ubuntu 20.04 版本，GCC 版本為 9.3.0，另外我用 __STDC_VERSION__這個 function 列出目前 C 的版本，可以看到結果為 201710，也就是 C18 的這個版本。

乙.

1. 執行 make，之後會產生四個執行檔案。請問你的執行節果為何？請附上畫面截圖

```
eric@eric-VivoBook-ASUSLaptop-X512FL-S512FL: ~/Desktop/OS/HW7
eric@eric-VivoBook-ASUSLaptop-X512FL-S512FL:~/Desktop/OS/HW7$ ./peterson_trival-g
p0: start
p1: start
進入次數 (每秒) p0: 8096572, p1: 8095017, 分別執行於 core#2 及 core#3
進入次數 (每秒) p0: 8178288, p1: 8178300, 分別執行於 core#2 及 core#3
進入次數 (每秒) p0: 8197218, p1: 8197210, 分別執行於 core#2 及 core#3
進入次數 (每秒) p0: 8168992, p1: 8168992, 分別執行於 core#6 及 core#3
進入次數 (每秒) p0: 8120403, p1: 8120353, 分別執行於 core#6 及 core#7
進入次數 (每秒) p0: 8154199, p1: 8154202, 分別執行於 core#6 及 core#7
進入次數 (每秒) p0: 8171193, p1: 8171193, 分別執行於 core#6 及 core#7
^C
eric@eric-VivoBook-ASUSLaptop-X512FL-S512FL:~/Desktop/OS/HW7$
```

```
eric@eric-VivoBook-ASUSLaptop-X512FL-S512FL:~/Desktop/OS/HW7$ ./peterson_trival-O3
p0: start
p1: start
進入次數 (每秒) p0:  1842, p1:     0, 分別執行於 core#1 及 core#0
進入次數 (每秒) p0:     0, p1:     0, 分別執行於 core#1 及 core#0
進入次數 (每秒) p0:     0, p1:     0, 分別執行於 core#1 及 core#0
進入次數 (每秒) p0:     0, p1:     0, 分別執行於 core#1 及 core#0
進入次數 (每秒) p0:     0, p1:     0, 分別執行於 core#1 及 core#0
進入次數 (每秒) p0:     0, p1:     0, 分別執行於 core#1 及 core#0
進入次數 (每秒) p0:     0, p1:     0, 分別執行於 core#1 及 core#0
^C
eric@eric-VivoBook-ASUSLaptop-X512FL-S512FL:~/Desktop/OS/HW7$
```

```
eric@eric-VivoBook-ASUSLaptop-X512FL-S512FL:~/Desktop/OS/HW7$ ./peterson_correct-g
start p0
start p1
進入次數 (每秒) p0: 4333904, p1: 4332649, 分別執行於 core#1 及 core#2
進入次數 (每秒) p0: 4384379, p1: 4375624, 分別執行於 core#1 及 core#2
進入次數 (每秒) p0: 4384158, p1: 4384109, 分別執行於 core#1 及 core#2
進入次數 (每秒) p0: 4386563, p1: 4377289, 分別執行於 core#5 及 core#6
進入次數 (每秒) p0: 4369896, p1: 4369901, 分別執行於 core#5 及 core#6
進入次數 (每秒) p0: 4356358, p1: 4355880, 分別執行於 core#5 及 core#6
進入次數 (每秒) p0: 4334554, p1: 4334654, 分別執行於 core#5 及 core#6
進入次數 (每秒) p0: 4348314, p1: 4348150, 分別執行於 core#5 及 core#2
進入次數 (每秒) p0: 4376112, p1: 4375110, 分別執行於 core#5 及 core#6
進入次數 (每秒) p0: 4368153, p1: 4368489, 分別執行於 core#5 及 core#6
^C
eric@eric-VivoBook-ASUSLaptop-X512FL-S512FL:~/Desktop/OS/HW7$
```

```
eric@eric-VivoBook-ASUSLaptop-X512FL-S512FL:~/Desktop/OS/HW7$ ./peterson_correct-O3
start p0
start p1
進入次數 (每秒) p0: 4210826, p1: 4180908, 分別執行於 core#1 及 core#4
進入次數 (每秒) p0: 4276486, p1: 4255368, 分別執行於 core#1 及 core#4
進入次數 (每秒) p0: 4273959, p1: 4250717, 分別執行於 core#1 及 core#4
進入次數 (每秒) p0: 4272482, p1: 4247125, 分別執行於 core#1 及 core#4
進入次數 (每秒) p0: 4269409, p1: 4245071, 分別執行於 core#1 及 core#4
進入次數 (每秒) p0: 4273707, p1: 4253803, 分別執行於 core#1 及 core#4
進入次數 (每秒) p0: 4286370, p1: 4264209, 分別執行於 core#1 及 core#4
進入次數 (每秒) p0: 4280237, p1: 4209072, 分別執行於 core#1 及 core#4
進入次數 (每秒) p0: 4282539, p1: 4256411, 分別執行於 core#1 及 core#4
進入次數 (每秒) p0: 4286218, p1: 4260607, 分別執行於 core#1 及 core#4
^C
eric@eric-VivoBook-ASUSLaptop-X512FL-S512FL:~/Desktop/OS/HW7$
```

2. 「確實的」解釋「為什麼」peterson_trival-03 的執行結果是錯的

```
eric@eric-VivoBook-ASUSLaptop-X512FL-S512FL: ~/Desktop/OS/HW7
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./peterson_trival-O3...
(No debugging symbols found in ./peterson_trival-O3)
(gdb) disass p0
Dump of assembler code for function p0:
   0x0000000000001360 <+0>:     endbr64
   0x0000000000001364 <+4>:     push   %rbx
   0x0000000000001365 <+5>:     lea    0xd13(%rip),%rdi        # 0x207f
   0x000000000000136c <+12>:    lea    0xced(%rip),%rbx        # 0x2060
   0x0000000000001373 <+19>:    callq  0x10e0 <puts@plt>
   0x0000000000001378 <+24>:    cmpl   $0x1,0x2cbd(%rip)        # 0x403c <flag1>
   0x000000000000137f <+31>:    movl   $0x1,0x2caf(%rip)        # 0x4038 <flag0>
   0x0000000000001389 <+41>:    movl   $0x1,0x2cad(%rip)        # 0x4040 <turn>
   0x0000000000001393 <+51>:    jne    0x13e4 <p0+132>
   0x0000000000001395 <+53>:    nopl   (%rax)
   0x0000000000001398 <+56>:    jmp    0x1398 <p0+56>
   0x000000000000139a <+58>:    nopw   0x0(%rax,%rax,1)
   0x00000000000013a0 <+64>:    mov    0x2c79(%rip),%rcx        # 0x4020 <stderr@@GLIBC_2.2.5>
   0x00000000000013a7 <+71>:    mov    $0x1e,%edx
```

可以看到在 24、31、41 這三行上，本來應該要先將 1 移到 flag0 跟 turn 中，再去執行 flag1 的 compare，但是 copiler 為了優化效能所以將 compare 移到前面，導致程式結果錯誤。

3. 請問在你的電腦上「peterson_trival-g」的速度比「peterson_correct-03」快或者是賣？上述二個程式的正確與否？

在我的電腦上，peterson_trival-g 跑的速度比 peterson_correct-03 還要快

```
Reading symbols from ./peterson_trival-g...
(gdb) disass /m p0
Dump of assembler code for function p0:
36      void p0(void) {
   0x00000000000012b7 <+0>:     endbr64
   0x00000000000012bb <+4>:     push   %rbp
   0x00000000000012bc <+5>:     mov    %rsp,%rbp

37          printf("p0: start\n");
   0x00000000000012bf <+8>:     lea    0xd9a(%rip),%rdi        # 0x2060
   0x00000000000012c6 <+15>:    callq  0x10e0 <puts@plt>

38          while (1) {
39              //🐉🐲 🌵 🌲 🌲 🌳 🌴 🌱 🌿 🍀
40                  //Peteron's solution的進去部分的程式碼
41              flag0 = 1;
   0x00000000000012cb <+20>:    movl   $0x1,0x2d5f(%rip)        # 0x4034 <flag0>
   0x0000000000001363 <+172>:   jmpq   0x12cb <p0+20>

42              turn = 1;
   0x00000000000012d5 <+30>:    movl   $0x1,0x2d4d(%rip)        # 0x402c <turn>

43              while (flag1==1 && turn==1)
   0x00000000000012df <+40>:    nop
   0x00000000000012e0 <+41>:    mov    0x2d4a(%rip),%eax        # 0x4030 <flag1>
   0x00000000000012e6 <+47>:    cmp    $0x1,%eax
   0x00000000000012e9 <+50>:    jne    0x12f6 <p0+63>
   0x00000000000012eb <+52>:    mov    0x2d3b(%rip),%eax        # 0x402c <turn>
   0x00000000000012f1 <+58>:    cmp    $0x1,%eax
   0x00000000000012f4 <+61>:    je     0x12e0 <p0+41>

44                  ;   //waiting
45
46
```

```
Reading symbols from ./peterson_correct-O3...
(No debugging symbols found in ./peterson_correct-O3)
(gdb) disass p0
Dump of assembler code for function p0:
   0x0000000000001370 <+0>:     endbr64
   0x0000000000001374 <+4>:     push   %rax
   0x0000000000001375 <+5>:     pop    %rax
   0x0000000000001376 <+6>:     lea    0xd02(%rip),%rdi        # 0x207f
   0x000000000000137d <+13>:    sub    $0x8,%rsp
   0x0000000000001381 <+17>:    callq  0x10e0 <puts@plt>
   0x0000000000001386 <+22>:    nopw   %cs:0x0(%rax,%rax,1)
   0x0000000000001390 <+32>:    movl   $0x1,0x2ca6(%rip)        # 0x4040 <flag>
   0x000000000000139a <+42>:    mfence
   0x000000000000139d <+45>:    mfence
   0x00000000000013a0 <+48>:    movl   $0x1,0x2c9e(%rip)        # 0x4048 <turn>
   0x00000000000013aa <+58>:    mfence
   0x00000000000013ad <+61>:    jmp    0x13bb <p0+75>
   0x00000000000013af <+63>:    nop
   0x00000000000013b0 <+64>:    mov    0x2c92(%rip),%eax        # 0x4048 <turn>
   0x00000000000013b6 <+70>:    cmp    $0x1,%eax
   0x00000000000013b9 <+73>:    jne    0x13c5 <p0+85>
   0x00000000000013bb <+75>:    mov    0x2c83(%rip),%eax        # 0x4044 <flag+4>
   0x00000000000013c1 <+81>:    test   %eax,%eax
   0x00000000000013c3 <+83>:    jne    0x13b0 <p0+64>
   0x00000000000013c5 <+85>:    callq  0x1140 <sched_getcpu@plt>
   0x00000000000013ca <+90>:    mov    %eax,0x2c84(%rip)        # 0x4054 <cpu_p0>
```

可以看到無論是 peterson_trival-g 或是 peterson_correct-O3，他們在執行的時候沒有為了效能把順序調換，因此可以正確地執行判斷進入 cs 的任務

4. 請「確實的」解釋「題三」，某個程式比另一個程式快或者慢的理由。

peterson_trival-g 沒有保證 mutual exclution，因此可能會有 p0 及 p1 同時進入的情況，雖然最終執行速度較快，但卻可能是錯的;相反的，peterson_correct-O3 雖然執行的慢，但他保證了 mutual exclution，因此執行的結果會是對的。