

工作內容：

工廠端系統維護(不用 24h on call)

● TSID (Technical System Integration 技術系統整合處) 最雷

1. 對工廠
2. 寫 java 或 c/c++
3. 進行產品驗證或針對工廠產線的程式
4. web service
5. 為了提高 wafer 的良率，工廠端會利用 IT 的 web service 出報告，利用這些報告分析，看哪邊還可以改善 wafer 的良率
6. 一般的工程師，負責系統開發(主要語言是 JAVA)，需要 on call。
主管是 TSID 下的開發科，所以主要是系統開發，分成三大塊：
 - ◆ 機台、設備自動化的系統，支援晶圓生產系統之類的
 - ◆ 在生產線上監控機台的系統
 - ◆ 機器學習、人工智慧 AI 的應用主管說新人前 3-4 個月會被派去做 1 的事情，之後視狀況轉到 2 或 3。
7. 工廠自動化、工程資料分析、設計研發的整合平台

● BSID (Business System Integration 企業系統整合處) 最爽

1. BSID 分成 2 大類別，共 5 個 team，主要在 2 廠
 - e-commerce => customer, vendor
 - e-business => supply chain, sales, 機動組
2. 電子商務系統、企業電子化
3. 對公司或人資招募人才，依需求去寫針對這些需求的程式

● ICSD (Infrastructure Communication Service 資訊建構暨通訊服務處)

1. 偏資安、巨資
2. 負責工廠基礎建設的維護，包含軟硬體、網路，有 error 時需盡快排除，工作重複性很高。
3. 關於網路等設施維修維護
4. 系統的軟硬體架構設計、整合與管理

可能被問的問題：

1. 基本自我介紹

- ◆ 名字→年紀→興趣(運動類、打球、重訓)→人格特質(樂觀外向、做事有規劃)→畢業學校→當兵→大學先修→優秀學生獎學金&書卷獎(大三成績明顯上升)→社團活動(班代、COMM5、幹訓、LAB 代表、所運)→大學專題→碩論
- ◆ 大家好，我是林祐宇，今年 25 歲，平常的興趣都比較偏向運動類，像是打籃球或重訓，我的個性非常樂觀外向，做事喜歡有計劃的做，大學畢業於國立中央大學通訊工程學系，碩士畢業於國立清華大學通訊工程研究所，4 月中剛退伍，所以現在才開始找工作，剛剛有提到，我做事喜歡有計劃的做，在一大二時，我就決定，大三結束以前要完成大學修課的畢業門檻，這樣在大四時，我就可以比別人早開始學研究所的課程內容，上研究所後，修課壓力會比同儕小，就可以花更多的心力在研究論文上，其中，在大三時，都是通訊相關的專業科目，我的成績也是在大三明顯上升，因此在大三的時候我還榮獲由中央大學校長親自頒獎的優秀學生獎學金，而且大四時也獲得書卷獎，接下來簡述一下我求學生涯曾參與過的重要活動，我參與過很多大型活動，這邊我挑幾個來說，第一個是班級代表，大學四年我被推選成為系上的代表，負責處理系上與學生之間的所有問題；第二個是五天四夜暑期營隊的執行長，負責規劃 5 天的所有行程、連絡校方、廠商等；第三個是幹部訓練，大學四年以來，我參加過兩場大型的幹部訓練，擁有一些基本的領導能力；第四個是碩士的實驗室代表，負責幫指導教授過濾人選，最後跟教授討論誰適合加入我們實驗室；最後一個是通訊所運動會總召，是一個所有學生及教授一起參加的一日運動會；最後，我的大學專題和碩士論文都做影像相關的題目，大學的專題是文字辨識暨翻譯系統，碩士論文是利用有限元素之多小波消除影像雜訊，以上是我的自我介紹，謝謝大家。

2. 碩論簡說

- ◆ 消除雜訊才有精確解→和傅立葉、單小波、多小波比較→應用分兩大類→輸入來源 2 種(圖像影像、訊號)→用途分類 4 種(資料預處理、邊緣偵測、資料壓縮、模式辨認)→多小波定義、特性→多尺度函數→多小波函數→多濾波器(2 種預處理方法)→問題定義→消除影像雜訊原理→測量依據(時間、Corr、PSNR、MSE、SAM)→結論
- ◆ 消除雜訊在影像分析中一直占有非常重要的地位，如果我們對含有雜訊的資料做進一步的分析，並沒有辦法得到精確的結果，因此消除雜訊是一個非常重要的技術，首先，先將我碩論中的多小波和以下三個常見的消除雜訊工具做簡單的比較，第一個是傅立葉轉換，傅立葉轉換是以正弦及餘弦函數為基底的轉換，將訊號從時域轉到頻域，而多小波轉換是時域轉時域的轉換，因此相對於傅立葉轉換，多小波轉換後可以在時域中得到許多傅立葉轉換後無法得到的資訊，第二個比較對象是單小波，單小波一次只能擁有一種特性，而多小波一次可以擁有許多特性，像是對稱性、反對稱性、正交性、短支撐等特性，這些特性在消除雜訊的過程中，可以提供我們更好的效能，最後比較的對象是多小波，目前大多數研究使用的多小波都是級數為 2 的多小波，其中最有名的是由 Geronimo-Hardin-Masopust(GHM)所創造的多小波，但 GHM 多小波的級數是固定為 2 的，因此消除雜訊的效能也有限，而我論文使用的多小波級數是可以自己決定的，在消除雜訊的過程可以有效的降低運算時間並提升正確率，是一個更能廣泛使用的多小波，而且，經過多小波消除雜訊的影像具有被壓縮的特性，對於大數據時代更是一大幫助。

- ◆ 多小波的應用也相當多元，我們可以依輸入來源分成兩大類，第一類是圖像及影像類，輸入訊號可以是人像、景色照、remote sensing image 等，第二類是訊號類，可以將人聲、音樂、機械聲、各種生醫訊號當輸入來源；如果我們依用途來分類，可以分成四大類，第一類是資料預處理，包含去除雜訊、使訊號平滑等，當未來要更進一步辨識時可以有更精確的結果，第二類是邊緣針測，剷除大部份的資料，僅保留影像的結構性，第三類是資料壓縮，輸入訊號每經過一次多小波轉換，維度都會降低一半，第四類是模式辨認，包含指紋辨識、靜脈辨識等。
- ◆ 介紹完應用後，開始介紹什麼是多小波轉換，多小波轉換是指用有限長或快速衰退的母小波震盪波形的縮放和平移來表示訊號。多小波有幾個比較重要的特性，第一個是短支撐，支撐長度如果越長，需要耗費更多的計算時間，如果太短，會不利於訊號能量的集中；第二個是對稱性和反對稱性，可以避免相位畸變；第三個是正交性，可以提高運算效能，並改善影像重建後的平方誤差值；多小波當然還有很多特性，但並不是擁有越多特性越好，而是要依照不同的輸入資料和目的來選擇適合的多小波。
- ◆ 多小波由多尺度函數和多小波函數所組成的，如果多小波的級數為 n ，代表多尺度函數是由 n 個尺度函數所組成的向量，要符合 dilation equation，多小波函數是由 n 個小波函數所組成的向量，要符合 wavelet equation，這兩條 equation 中會有 3 個低通濾波器的係數 C_k 和 5 個高通濾波器的係數 D_k ，這些濾波器係數在我的碩論中有 closed-form 可以直接獲得，因為 C_k 和 D_k 都是 $n \times n$ 的矩陣，每個 C_k 和 D_k 的列向量都要對應到一個數據流，因此我們要把輸入訊號重複 n 次，目前常見的方法有兩種，第一個方法是 repeat row，將輸入訊號依照不同 weighting 重複，適合用在特徵擷取，重點訊號也會因為重複而被加強，但不適合用在資料壓縮，因為訊號重複時，殘餘項也被重複了，會降低壓縮的效能；第二個方法是 Critical sampled scheme，將輸入訊號經過採樣後分不到不同的列向量，如此一來不會有過多的殘餘項，適合用在資料壓縮的預處理，如果用在特徵擷取將會增加整體運算時間。
- ◆ 雜訊在頻率分佈上，主要出現於高頻居多，利用多小波轉換的多濾波器將訊號高頻及低頻分離，先取出低頻成份，再經過適當的整理與排列後，將每個數據流中的第一項抓出來，便可得到消除雜訊後的影像，且此影像也有被壓縮過。
- ◆ 介紹完工具後，我來定義一下問題，我的問題是 將含有雜訊的影像，利用各式測量標準，將雜訊極小化，求得最好的效能。此次的測量標準我選擇時間、相關係數、PSNR、MSE、SAM 當標準，時間越短越好，相關係數越靠近 1 表示消除雜訊效果越好，PSNR 越大表示 spatial quality 越好，MSE 越小表示 global quality 越好，SAM 越小表示 spectral quality 越好。
- ◆ 影像選擇的部份，除了影像分析中最常使用的 Lena 圖之外，我還選擇 remote sensing 影像，然後再挑兩個特別的影像，清華大學的校花和我的系館空拍照，目的是為了讓大家知道，利用多小波消除雜訊的影像並沒有限制某幾種影像而已。最後實驗的結果是，如果雜訊較小時，使用級數較高的有限元素之多小波；若雜訊較大時，依照效能或時間考量，使用不同級數的有限元素之多小波，如此選擇不但可以降低運算時間，又可以得到不錯的濾波結果。
- ◆ 以上是我的碩士論文簡說，謝謝大家。

3. 你覺得最成功的一件事？

- ◆ 做人、個性、樂觀

4. 你覺得最失敗的一件事，學到什麼，如何改善？

- ◆ 大一大二花太多心力在活動上，沒認真讀書，大三大四努力改善
- 5. 台積電公司責任
 - ◆ 願景：提升社會
 - ◆ 使命：誠信正直、強化環保、關懷弱勢
- 6. 為什麼想來我們公司？
 - ◆ TSMC 幫台灣賺了不少錢，我希望我的工作，除了是能養活自己，也能對國家經濟有幫助
 - ◆ TSMC 在專業晶圓代工上是頂尖企業，要求進步就要不斷接觸新技術，我在 TSMC 有機會接觸並應用最新技術
 - ◆ 穩定，薪資福利優於其他公司
- 7. 合作經驗 (分活動 & 讀書方面)
 - ◆ 大學辦營隊，需要跟不同系的人和教職員合作
 - ◆ 實驗室一直都有外籍生和大陸生，他們受的教育訓練和我們台灣人有些差異，所以每次有新知識要學，我們會分別讀完一次，再一起討論哪些細節是彼此沒有注意到的
- 8. 遇到困難如何解決
 - ◆ 先思考為什麼會有這個問題，拿出紙筆寫下來，再檢視自己的過程哪裡出問題，逐一抽絲剝繭，再思考有幾種解決辦法，最後挑出一種最適當的方法來解決問題
- 9. 妳的缺點
 - ◆ 有時為了力求完美反而變吹毛求疵
 - ◆ 過度樂觀
- 10. 妳的優點
 - ◆ 樂觀外向，很容易融入新的環境
 - ◆ 做事習慣規劃完整，有邏輯、條理的做
 - ◆ 善於溝通及表達自己的想法
 - ◆ 做事負責，態度積極，具領導能力
- 11. 遇到問題時的解決方式、心態如何調整
 - ◆ 最重要的是解決問題，不追究責任
- 12. 社會新鮮人會問對工作展望未來期許
 - ◆ 希望進公司後能努力學習，並在最短時間內學會職務上應具備之各項技能；中期則希望能提升專業能力並朝管理階層邁進；未來則期望自己成為一位能夠獨當一面的專業經理人
- 13. 求學期間最有興趣的學科
 - ◆ 通訊之最佳化方法
 - ◆ 可以依照需求來調配參數，得到最佳化的結果，或是最省成本的結果
- 14. 求學期間最差的學科
 - ◆ 程式語言、資料結構
 - ◆ 因為大一時間分配上出了些問題，導致程式語言基礎沒學好，所以資料結構較弱
 - ◆ 大三大四後有開始補強
- 15. 現場的人不好相處，遇到衝突如何處理
 - ◆ 因為是新人，只要資深員工有任何批評，基本上都是想讓新人更進步，更快上軌道，虛心受教就好
- 16. 什麼情況下你會離開公司

- ◆ 我會一直待到公司請我離開才離開，不然不走
17. 你寫過最大的程式專案是什麼？中間最難的部份是哪裡？你如何克服這個難題？
- ◆ 碩論程式
 - ◆ 輸出影像具有壓縮性，原始輸入的影像會被縮小且複製四次呈現，但其中只有部分影像是我需要的結果，因此，我需要依照需求挑出我要的影像
 - ◆ 重新回到理論，仔細推導出不同性質影像的位置，再實現到程式中
18. 回想一下哪些時候修過什麼課、課程內容大概是什麼
- ◆ 最佳化理論
 - ◆ 如何將問題以數學式子表示，並算出對應參數為多少時可得到最佳解，從中也可以獲得參數設定為多少時，解出來的結果會越好或越差
 - ◆ 研究和工作最大的不同在於，做研究時，通常我們都要得到最好的答案、最佳的參數，讓我們的效能最好，不太需要去考慮現實因素；但工作時，有時我們只需要符合顧客要求的門檻就好，不需要做到最好的效能，畢竟公司的目標還是以獲利為主
 - ◆ 舉例來說，電信業者，基地台越多手機的訊號就會越好，但是電信業者不需要讓每個人手機訊號最好，他只需要維持大部份的顧客都可以正常使用手機的訊號就好，因此我們需要計算出，大約多遠要架一個基地台
19. 假設時間內給你的任務做不完你會怎麼處理？
- ◆ 先向主管報告任務無法時限內完成，並向主管說明原因、以及再多久可以完成，如果是任務內容有些困難無法克服，也會和主管報告清楚
20. 問問題所有人都剛好在忙怎麼辦？
- ◆ 我習慣將自己的問題用文字重新敘述一次，因為這樣可以讓我重新理解一次問題，所以如果遇到問問題所有人都剛好在忙的時候，我會將我的問題以文字方式留言給我求助的人，並在之後適當的時間點再去找他一次
21. 想在哪個地點工作？為什麼？
- ◆ 新竹
 - ◆ 生活上，因為研究所就在園區旁邊讀書，所以也很習慣這邊的生活和飲食
 - ◆ 知識上，因為之前的朋友和學長姐也幾乎在園區工作，未來如果有什麼問題要請教，在新竹也最方便
22. 台積有規定不能帶手機入廠，假設看到朋友拿出了手機你會怎麼做？
- ◆ 先去和那個朋友了解為什麼會這麼做，如果有什麼困難或許可以大家一起幫忙，如果是刻意這麼做，我會先告知他不能這麼做，如果再有下次，一定會告訴主管
23. 假設團隊裡的大家都很忙，每個人都有自己的任務要完成，現在突然又多了一樣任務你會怎麼做？
- ◆ 先評估這項任務大約花妳多少的時間，如果可以短時間內解決，可以在進行原任務的同時，分一些時間做多的任務；若短時間內無法解決，先完成手邊的原任務，再做新任務，當然還要考慮新舊兩個任務的迫切性
24. 跟指導教授相處的模式？
- ◆ 研究上，指導教授不喜歡直接告訴我解決方式，喜歡用引導的方式告訴我「為什麼有這些問題？該如何解決？」，希望我能多思考，畢竟做研究和寫作業最大的差別在於「有無正確答案」，教授要讓我知道，研究沒有結束的一天，只有這個答案完不完整，當然在我絞盡腦汁都想不到解決方法時，教授也會適當的給我一些提示，讓我多去 survey

- ◆ 私底下，因為教授的個性比較開明，與其說是指導教授，他更像我的親哥哥，除了研究上的問題，我們也經常討論重訓、籃球等話題，甚至還會一起吃飯、慶祝等等，如同亦師亦友那樣。
- 25. 有指導教授交待的進度沒完成的經驗嗎?怎麼處理?
 - ◆ 因為我的指導教授個性和大多數教授並不同，所以教授不會主動要求我在什麼時間點內要完成什麼事情，教授指會告訴我，哪些事情是我該做的，你要什麼時候做、做幾分都由你自己決定，所以在研究所的這兩年中，我的自主管理能力也提升不少。
- 26. 有遇過甚麼比較重大的挫折? 難過程度幾分(1~10)? 怎麼紓解壓力? 紓解完後難過程度變幾分?
 - ◆ 大二時，擔任 5 天 4 夜暑期營隊的執行長，當時和其他組長意見有分歧，當時紓解壓力的方法是找幾個朋友一起出去走走，多聊一點開心的事情，心情自然就恢復了
- 27. 遇到挫折後如果別人找你討論事情，你會如何跟他討論?
 - ◆ 如果是討論讓我受挫的事情，我會靜下心來，先聽聽他的想法，或許可以讓我心情好一些
 - ◆ 如果是討論和受挫無關的事情，那就更應該好好處理，不要把情緒帶入工作
- 28. 團隊合作經驗? 有沒有意見衝突?
 - ◆ 大二時，擔任 5 天 4 夜暑期營隊的執行長，當時和其他組長意見有分歧，當天開會結束後，我有找其他營隊的執行長一起聊聊，聊完後再和自己底下的組長們重新溝通一次
- 29. 大學學科有哪科必修或相關不好，會被問。(會被問為什麼不好)
 - ◆ C 語言，因為剛上大學的時候玩心比較重，所以有點荒廢了程式語言的部份，但研究所之後有努力重新學過一次
- 30.

要問公司的問題：

1. 詳細工作內容
2. 新進員工的職業培訓
3. 公司氣氛、環境
4. 平均上班時數
5. 月薪、保障幾個月
6. 是否出差

要準備的科目：

1. DSP (寶基書)
2. 通原 (魏瑞益講義)
3. C 語言 (鍊結、記憶體配置、int char 幾個 byte)
4. 計組 (陳彥文講義) (看看 PIPELINE、sram dram 快取)
5. 資結 (許憲聰講義)
6. ~~Convex (茄子書)~~
7. ~~多小波 (碩論)~~
8. 無線通訊 (蔡育仁講義)
9. ~~ADMM 相關論文~~
10. 多益
11. 通訊系統的 block 圖 每個 block 解釋
12. 各種 fading 的種類
13. 肥臉筆記

要面試的公司：

- 台北：
 1. 和碩
 2. 華碩
 3. 緯穎
 4. 南亞科
 5. 仁寶
 6. 華新科
 7. 威盛
 8. DELL
 9. 聯想
- 桃園：
 1. 廣達
 2. 台達電
 3. 美光
 4. 景碩
 5. 中科院
- 新竹：
 1. 合勤

2. 明泰
 3. 工研院
 4. 矽創
 5. 鴻海
 6. 聯電
- 真正要去的公司：
 1. 台積電 (晶圓代工)
 2. 友達 (晶圓代工)
 3. 聯發科 (IC)
 4. 新思 (IC)
 5. 慧榮 (IC)
 6. 創意 (IC)
 7. 聯詠 (IC)
 8. 群聯 (IC)
 9. 益華 (IC)
 10. 絡達 (IC)
 11. 晨星 (IC)
 12. 擎亞 (IC)
 13. 信驊 (IC)
 14. 敦泰 (IC)
 15. 瑞昱 (藍芽 wifi)
 16. 啟碁 (軟韌體)
 17. 原相 (影像處理)
 18. 安霸 (影像處理)
 - 分類說明
 1. 已丟履歷
 2. 已面試

已面試的公司：

- 台北：
 1. 睿緻科技 4/30(二)
考 C 手寫 10 題，沒問專業、沒給一段時間自介&講碩論，都以聊天方式進行

Note: This test will take about 30~45 minutes. Please don't access internet during the test.

1. What does the following program print?

```
#include<stdio.h>
int main()
{
    int a = 5, b;
    a >= 10 ? b=100: b=200;
    printf("%d\n", b);
    return 0;
}
```

Ans.

2. Point out errors in the program

```
f(int a, int b)
{
    int a;
    a = 20;
    return a;
}
```

3. What will be the output of the following C code?

```
#include <stdio.h>
main()
{
    int arr[]={6, 16, 26, 36, 46};
    int *ptr = arr;
    *(ptr) += 4;
    printf( "%d\n", *ptr);
    *(ptr++) += 4;
    printf( "%d\n", *ptr);
    *(ptr) += 4;
    printf( "%d\n ", *ptr);
    printf( "%d,%d\n ",*ptr, *(++ptr));
}
```

Ans.

第一題： 答案是 200

If 用三元運算子寫 $a \geq 10 ? b=100: b=200$; 如果 $a \geq 10$ 成立則 $b=100$; 不成立 $b=200$

$\text{int } i=5; a = i > 0 ? 2;$ 輸出 $a=1$ (表示 true) (比較運算的，沒打 if 成立輸出，就用 $\text{true}=1$)

$\text{int } i=5; a = i ? 2;$ 輸出 $a=5$ (只要 i 不是 0，表示 true， a 直接輸出 i 的值)

$\text{int } i=-5; a = i ? 2;$ 輸出 $a=-5$ (不管 i 的正負，只要不是 0，都是 true)

$\text{int } i=0; a = i ? 2;$ 輸出 $a=2$ ($i=0$ 表示 false， a 輸出為 else 答案， $a=2$)

(直接是一個數字的，true 輸出即該數字；false 輸出即 else 答案)

(只可以不打 true 結果，else 結果一定要打)

(在 C 語言中，非零的數都是 true，只有 0 是 false)

第三題： 答案是 10 16 20 26 26

$*ptr = *(ptr)$ 都是印值

印地址用 $\%p$

$*(ptr++) += 4$ 即 $*(ptr++) = *(ptr++) + 4$ 先用原地址求值，再移動”一次”地址

```

int main()
{
    int arr[]={6,16,26,36,46};
    int *ptr = arr;

    *(ptr) += 4;
    printf("%d\n", *ptr); 10

    *(ptr++) += 4;
    printf("%d\n", *ptr); 14,16,26,36,46

    *(ptr) += 4;
    printf("%d\n", *ptr); 20 14,20,26,36,46

    printf("%d,%d,%d\n", *ptr, *(++ptr), *(ptr++));
}

```

紅色粗體表示pointer目前位置
printf從右往左印

```

C:\Users\Administ
10
16
20
36,36,20
Process returned
Press any key to

```

```

int main()
{
    int arr[]={6,16,26,36,46};
    int *ptr = arr;
    printf("ptr = %x\n\n", ptr);

    printf("*ptr = %d\n", *ptr);
    printf("*ptr = %d\n", *ptr);
    printf("++*ptr = %d\n", ++*ptr);
    printf("++*ptr = %d\n", ++*ptr);
    printf("*ptr = %d\n\n", *ptr);

    printf("** (ptr) = %d\n", * (ptr));
    printf("** (ptr++) = %d\n", * (ptr++));
    printf("** (ptr) = %d\n", * (ptr));
    printf("** (ptr)++ = %d\n", * (ptr)++);
    printf("** (ptr) = %d\n", * (ptr));
    printf("**++ (ptr) = %d\n", **++ (ptr));
    printf("** (ptr) = %d\n", * (ptr));
    printf("++* (ptr) = %d\n", ++* (ptr));
    printf("** (ptr) = %d\n\n", * (ptr));
}

```

Printf 會改陣列裡的值

```

C:\Users\Administ
ptr = 28ff08

*ptr = 6
*ptr = 6
++*ptr = 7
++*ptr = 8
*ptr = 8

*(ptr) = 8
*(ptr++) = 8
*(ptr) = 16
*(ptr)++ = 16
*(ptr) = 26
++*(ptr) = 36
*(ptr) = 36
++*(ptr) = 37
*(ptr) = 37

Process returned
Press any key to

```

4. Write down a function which fulfill following condition
 input: 0~7 return 0
 input: 8~15 return 8
 input: 16~23 return 16
 input: 24~31 return 24

5. The following code implements bubble-sorting algorithm, but there are some mistakes in the code. Can you correct them?

```
void bubble_sort(void)
{
    int data[5] = {5, 8, 7, 9, 0};
    int i, j;

    for (i=0; i < sizeof(data); i++) {
        for (j = sizeof(data); j > i; j--) {
            if (data[j-1] > data[i]) {

                data[j-1] = data[i];
                data[i] = data[j-1];
            }
        }
    }
    printf("\nBubble-sorting results:\n");
    for (i = 0; i < 5; i++) {
        printf("%d ", data[i]);
    }
}
```

6. Please explain the roles of bootlader, kernel and filesystem, in an embedded system.

7. What is DMA(Direct memory access)? What is the benefit of DMA?

8. Please explain "volatile" in C/C+ language.

9. List 10 Linux shell commands and describe their usages

10. Explain IPC (Inter-process communication) and list at least 3 methods implemented in Linux IPC facilities.

第七題：直接記憶體存取（Direct Memory Access，DMA）一種記憶體存取技術。就是搬資料，不需要用到 CPU 的指令。只需要 1.要知道原始資料的位置。2.要知道目的地位置。3.要知道要搬多少資料長度。

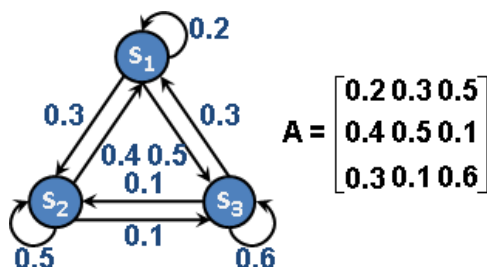
第八題：volatile 關鍵字可以用來提醒編譯器它後面所定義的變量隨時有可能改變，因此編譯後的程序每次需要存儲或讀取這個變量的時候，都會直接從變量地址中讀取數據。“直接存取原始記憶體地址”（下面還有解說）

2. 雲守護 5/7(二) 沒談薪水 不會保證幾個月 不用出差 台灣公司 (面 2H)
有自介(漏講社團)，問學的好的科目(convex、排隊)，有續問
機率問題、傅立葉轉換&分析、排隊舉例、馬可夫鍊、global 變數+static 差在哪、process
邏輯 3 題、交換程式

$\int_{-\infty}^{\infty} e^{iw_2t} e^{-iw_1t} dt$ 積分後是 delta func.，當 $w_2 = w_1$ 時，得到 1，(t=0 為無限大，積分後是

1)，表示正交

馬可夫鍊(Markov chain)是一個狀態 x 到另一個狀態 y 的隨機過程， $P(y|x)$ 表示，轉移的機率總和為 1，轉移機續習慣以矩陣表示，稱「轉移矩陣 transition matrix」，具備「無記憶」的性質：下一狀態的機率分布只能由當前狀態決定，在時間序列中它前面的事件均與之無關。一個狀態的抵達機率：所有來源狀態各自乘上轉移機率，再加總。走 ∞ 步，可能匯合到某些狀態，也可能循環繞圈。



排隊理論 A/B/C/X/Y，A 表示到達規則，B 表示服務規則，C 表示服務台個數，X 表示系統容量，Y 表示等後規則(先來先服務、後來先服務、隨機選擇服務、由優先權決定、通用規則)，常見有 M/M/1、M/M/2、M/M/s、M/M/1/K、M/G/1。分佈最常見的是 Poisson 分

$$P(X = k) = \frac{e^{-\lambda} \lambda^k}{k!}$$

佈， λ 表示單位時間內到達規則(或事件發生機率，為已知)

←Poisson 分佈圖

- ①系統負荷水平 ρ ：它是衡量服務台在承擔服務和滿足需要方面能力的尺度；
- ②系統空間機率 P_0 ：系統處於沒有顧客來到要求服務的概率；
- ③隊長：系統中排隊等待服務和正在服務的顧客總數，其平均值記為 L_s ；
- ④隊列長：系統中排隊等待服務的顧客數，其平均值記為 L_g ；
- ⑤逗留時間：一個顧客在系統中停留時間，包括等待時間和服務時間，其平均值記為 W_s ；
- ⑥等待時間：一個顧客在系統中排隊等待時間,其平均值記為 W_g 。M/M/1 排隊系統是一種最簡單的排隊系統。系統的各项指標可由圖 2 中狀態轉移速度圖推算出來(表 1)。

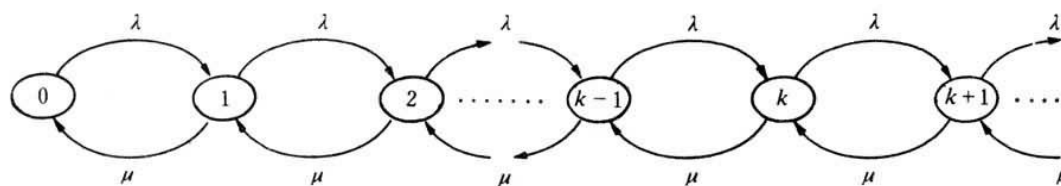


图 2 状态转移速度图

表 1 M/M/1 排队系统的指标

指标	ρ	P_0	L_s	L_g	W_s	W_g
计算公式	$\frac{\lambda}{\mu}$	$1 - \frac{\lambda}{\mu}$	$\frac{\lambda}{\mu - \lambda}$	$\frac{\lambda^2}{\mu(\mu - \lambda)}$	$\frac{1}{\mu - \lambda}$	$\frac{\lambda}{\mu(\mu - \lambda)}$

3. 和碩 5/10(五) 沒談薪水 出差可不去 會有二面

有自介，碩論簡說，聊天居多，考邏輯、性向測驗，等三個多小時才輪到我

- 4.
- 桃園：
 - 1.
- 新竹：
 - 1. 鴻海 4/29(一) 56K get 保 14 滿一年後保 16 (面 3H)
英文+性向測驗，沒問專業、沒給一段時間自介&講碩論，都以聊天方式進行
 - 2. 聯發科 5/9(四)Tony 一面 5/13(一)FAE 一面+Tony 二面
上機考 40 分，筆試 40 分 (題目和網路考古完全不同)
白板題考 Stack 的 pop push function 實做，進階問如何用動態記憶體 malloc 寫
 - 3. 明泰 5/14(二) 沒談薪水 不太加班 考英文
沒問專業 聊天居多

知識：

- 線上編譯器
https://www.tutorialspoint.com/compile_c_online.php
- 一堆考古題分享
<https://www.ptt.cc/bbs/NTUE-CS100/M.1300374249.A.C8F.html>
<https://knowledgemyfriend.blogspot.com/2016/10/whats-highlight.html>
- Tony 哥給的網站
<http://linux.vbird.org/>
<https://git-scm.com/book/zh-tw/v2>
- SDN 軟體定義網路 (Software Defined Networking)
https://www.netadmin.com.tw/article_content.aspx?sn=1603310002
<https://medium.com/@RiverChan/sdn%E8%88%87nf%E7%9A%84%E5%8D%80%E5%88%A5%E5%92%8C%E9%97%9C%E4%BF%82-3a15692bb3f6>
- static 靜態
使用在全域變數或全域函式 (Global variable & Global function)
讓該變數(或該函式)的可視範圍只侷限在該檔案內，其他的 .c 檔看不到此變數(或函式)的存在 (可以用來限定變數的作用域)。即使其他檔案用 extern 宣告也看不到！把 Global 的變數或函數變成了「internal linkage 內部連結」，當 Linker 在找 symbol 時是會忽略它的。

使用時機：當此全域變數(或全域函式)不想被其他檔案引用和修改時，或者不同檔案可以使用相同名字的全域變數(或全域函式)而不會產生命名衝突。

使用在函式內的區域變數 (Local variable)

因為區域變數預設就是動態變數，而在區域變數前加上 static 修飾字則會將變數由動態(dynamic)變數轉為靜態(static)變數，靜態變數的壽命(lifetime)與動態變數不同，靜態變數會一直存在(生命週期類似全域性變數，但是作用域不變)，直到程式結束為止(靜態局部變數只能初始化一次)。但由於它是被宣告在函式之中，所以函式之外仍無法存取 static 變數。

使用時機：在函式當中，我們經常會需要用到上一次執行的結果，如果不使用 static，就必須使用全域變數。 ex：統計次數的功能(某函式被呼叫幾次)。

<https://openhome.cc/Gossip/CGossip/Scope.html>

<http://archerworks.blogspot.com/2010/07/cstatic.html>

- extern 外部的 讓不同檔案間可以共用同一個變數，此變數宣告必須是”全域變數”

若使用 gcc some.c main.c 進行編譯

some.c 有以下內容

```
double some_var=1000;
```

main.c 有以下內容

```
int main(){
extern double some_var;    //extern 指出 some_var 是在其他位置被定義，編譯器會試圖在其它
                           位置或文件中找出 some_var 的定義，結果在 some.c 中找到
printf(“%f”,some_var);    //印出 1000.000000
}
```

extern 聲明 some_var 在其他位置被定義，如果要改 some_var 的值，必須這麼做

main.c 有以下內容

```
int main(){
extern double some_var;    //extern 指出 some_var 是在其他位置被定義，編譯器會試圖在其它
                           位置或文件中找出 some_var 的定義，結果在 some.c 中找到

some_var=99;
printf(“%f”,some_var);    //印出 99.000000
}
```

(不可以直接 extern double some_var=99; 會引發重複定義錯誤)

<https://openhome.cc/Gossip/CGossip/Scope.html>

<https://codertw.com/ios/326890/?fbclid=IwAR2p8N2AE1ONgeUPPJc997jdTITCo2YykPFYXqbhKGpBgoRAqEXhplut4r4>

- const 常量 修飾後面的東西，被修飾的變數只讀，不能被修改
當*在 const 之前，指標本身為 const，為常數指標(int * const p)

當*在 const 之後，指到的資料型態為 const，為常數變數(const int *p = int const *p)
一個參數可以是 const 又是 volatile，宣告為 volatile 因為他可能被意想不到的修改掉，宣告為 const 因為我們的程式不能修改他的值(唯讀)

[https://lalalah.pixnet.net/blog/post/31677924-](https://lalalah.pixnet.net/blog/post/31677924-%5B%E8%BD%89%5D%E7%9A%84%E9%97%9C%E9%8D%B5%E5%AD%97%E2%80%94%E2%80%94const%E7%9A%84%E7%90%86%E8%A7%A3%E5%92%8C%E7%94%A8%E6%B3%95)

[%5B%E8%BD%89%5D%E7%9A%84%E9%97%9C%E9%8D%B5%E5%AD%97%E2%80%94%E2%80%94const%E7%9A%84%E7%90%86%E8%A7%A3%E5%92%8C%E7%94%A8%E6%B3%95](https://lalalah.pixnet.net/blog/post/31677924-%5B%E8%BD%89%5D%E7%9A%84%E9%97%9C%E9%8D%B5%E5%AD%97%E2%80%94%E2%80%94const%E7%9A%84%E7%90%86%E8%A7%A3%E5%92%8C%E7%94%A8%E6%B3%95)

- volatile 易變的 嵌入式系統常用

被 volatile 修飾的變數代表它的值有可能因為編譯器不知道的因素修改，
所以告訴編譯器不要對它涉及的地方做最佳化，
並在每次操作它的時候都去讀取該變數實體位址上最新的值，
而不是讀取 CPU 暫存器上的值，
一般的變數可能因為剛剛讀取過而放在 CPU 暫存器上使動作變快。

例子：

- (1) 硬體暫存器，如狀態暫存器。
- (2) 多執行緒所共用的全域變數。
- (3) 中斷服務函式 (Interrupt Service Routine, ISR)所使用的全域變數。

- **全域變數**是指直接宣告在（主）函式之外的變數，這個變數在整個程式之中都「看」得它的存在，而可以呼叫使用。

區域變數是指宣告在函式之內的變數，或是宣告在參數列之前的變數，它的可視範圍只在宣告它的函式區塊之中，其它的函式不可以使用該變數。

區塊變數是指宣告在某個陳述區塊之中的變數，例如 while 迴圈區塊中，或是 for 迴圈區塊。當可視範圍大的變數與可視範圍小的變數發生同名狀況時，可視範圍小的變數會暫時覆蓋可視範圍大的變數，稱之為「**變數覆蓋**」。

- typedef 幫資料型態取別名

typedef int hihi 之後打 hihi 即 int，hihi a = int a

struct var {}; typedef struct var haha;之後打 haha 即 struct var，struct var a = haha a

<https://zh.wikipedia.org/wiki/Typedef>

- pointer 「從最內層的括號讀起，變數名稱，然後往右，遇到括號就往左。當括號內的東西都解讀完畢了，就跳出括號繼續未完成的部份，重覆上面的步驟直到解讀完畢。」

丟值永遠不改 丟址才可以改 搞清楚現在丟值還是址 就很簡單了

- a) int a; 一個整型數 // An integer
- b) int *a; 一個指向整數的指標 // A pointer to an integer
- c) int **a; 一個指向指標的指標，它指向的指標是指向一個整型數 // A pointer to a pointer to an integer
- d) int a[10]; 一個有 10 個整數型的陣列 // An array of 10 integers
- e) int *a[10]; 一個有 10 個指標的陣列，該指標是指向一個整數型的 // An array of 10 pointers to integers

f) `int (*a)[10];` 一個指向有 10 個整數型陣列的指標 // A pointer to an array of 10 integers
g) `int (*a)(int);` 一個指向函數的指標，該函數有一個整數型參數並返回一個整數 // A pointer to a function a that takes an integer argument and returns an integer
h) `int (*a[10])(int);` 一個有 10 個指標的陣列，該指標指向一個函數，該函數有一個整數型參數並返回一個整數

`*p++` 後動址
`*(p++)` 後動址
`(*p)++` 後動值
`*++p` 先動址
`*(++p)` 先動址
`++*p` 先動值
`++(*p)` 先動值

```
Code
int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
int *p = &(a + 1)[3];
printf("%d\n", *p);
```

- 輸出 5，因為 `a+1` 指向 `a` 的第二個元素，`[3]` 表示再向後移動 3 個元素
- `a+1` 是跳一個 `int` 的大小 (`a` 想成指標)
- `&a+1` 是跳一整個 array 的大小
- `int var[3] = {10, 100, 200};`
 - `var` 是 constant 不能用 `var++` 只能 `*(var+2) = 200`

<http://hackgrass.blogspot.com/2018/03/c-pointerint-foo-int-bar.html>

http://edisonshih.pixnet.net/blog/post/27961904-02_%E9%99%A3%E5%88%97%E8%88%87%E6%8C%87%E6%A8%99%E9%97%9C%E4%BF%82

<https://kopu.chat/2017/05/15/c%E8%AA%9E%E8%A8%80-%E8%B6%85%E5%A5%BD%E6%87%82%E7%9A%84%E6%8C%87%E6%A8%99%EF%BC%8C%E5%88%9D%E5%AD%B8%E8%80%85%E8%AB%8B%E9%80%B2%EF%BD%9E/>

<https://dotblogs.com.tw/brian/2012/10/18/77588>

- 動態記憶體 `DataType *ptr = (DataType*) malloc(所需的記憶空間的 byts 數);`
`DataType*` 傳回的位址強制轉換成指標變數 (`DataType *ptr`) 所指向的型態
改變原大小 `ptr = realloc(ptr, 新的大小);`

擴充從後方新增更多的空間，原本存的資料會被保留，而後方新增的空間則是會處於未初始化的狀態；縮減從後方開始縮減，前段未被縮減空間中所儲存的資料也會被保留。(若將新大小改為 0，和 free 的功能一樣)

配置完空間後，回傳第一個 byte 的指標；若沒拿到記憶體空間，回傳 NULL，所以每次配置完最好檢查一下是否 ==NULL？

用完記憶體空間要 free()，不然會造成記憶體洩漏 (memory leak)。

(*) 利用 typedef 簡化資料的表示

```
struct StudentS
{
    char no[11+1];
    int chi;
    int eng;
    int math;
}; // 分號 ; 不可或缺!
```

使用法：(In C)

```
struct StudentS stu, *pstu;
// pstu 是指標。
```

```
typedef struct
{
    char no[11+1];
    int chi;
    int eng;
    int math;
} Student, *pStudent;
```

// Student, pStudent 現在都是自訂的資料型態。

使用法：

```
Student stu;
pStudent pstu; // pstu 是指標。
```

```
void genStudent(int N)
{
    // 有三種使用方式：
    Student *stu = (Student*) malloc(sizeof(Student) * N);
    // struct StudentS *stu = (Student*) malloc(sizeof(struct StudentS)*N);
    // pStudent stu = (pStudent) malloc(sizeof(Student) * N);

    if ( stu==NULL )
    {
        printf("Memory allocation failed!\n");
        return;
    }

    int k, c;
    randomize();
    for (k=0; k<N; ++k)
    {
        sprintf(stu[k].no, "S9917%03d", k+1 ); // 另一種寫法

        stu[k].chi = random(101);
        stu[k].eng = random(101);
        stu[k].math = random(101);

        printf("%s: %d, %d, %d.\n",
            stu[k].no, stu[k].chi, stu[k].eng, stu[k].math);
    }

    free(stu); // 有借要還！一定要記得釋放記憶體！
}
```

(*) sprintf (儲存輸出的字串變數，格式字串 [，參數列])；

sprintf(.) 的功能全同於 printf(.) 差別在：printf 會將結果送往 stdout (通常就是螢幕)；而 sprintf 會將相同的結果寫入字串變數中。不過要注意的是，用來儲存輸出的字串的空間不可太小以免資訊無法正確存入，導致訊息失真。

<https://blog.gtwang.org/programming/c-memory-functions-malloc-free/>

http://140.129.118.16/~richwang/99-2-Courses/About_C_DynamicMemoryAllocation.pdf

● 指標 和 陣列 差別

就記憶體方向來看，"指標"所用的記憶體位置不為連續，而"矩陣"所配置的空間為連續。

宣告一個陣列的同時，電腦就會配置與該陣列等量大小的記憶體供陣列使用。

指標就是一個地址，陣列名稱表示第一個元素地址。

陣列是指標的一種，但指標不是陣列的一種。

- 記憶體配置概念 堆疊(Stack)是後進先出的集合，佇列(Queue)是先進先出的方式處理物件集合
Stack 從高記憶體位置往低記憶體位置堆疊，Heap 從低往高堆疊

Stack(堆疊區)存區域變數、函數...

Heap 存動態配置的變數

Data segment(資料區)存全域變數、static 變數、常數

<https://blog.gtwang.org/programming/memory-layout-of-c-program/>

- and & or | xor ^ not~ (跟 1 補數~運算)

~ 就是跟 1 做補數，即跟 1 做 XOR，做出來結果即 not

[https://blog.xuite.net/tsai.oktomy/program/67226142-](https://blog.xuite.net/tsai.oktomy/program/67226142-%E4%BD%8D%E5%85%83%E9%81%8B%E7%AE%97%E5%AD%90+%26+AND%2C+%7C+OR%2C+%5E+XOR%2C+~+NOT+)

[%E4%BD%8D%E5%85%83%E9%81%8B%E7%AE%97%E5%AD%90+%26+AND%2C+%7C+OR%2C+%5E+XOR%2C+~+NOT+](https://blog.xuite.net/tsai.oktomy/program/67226142-%E4%BD%8D%E5%85%83%E9%81%8B%E7%AE%97%E5%AD%90+%26+AND%2C+%7C+OR%2C+%5E+XOR%2C+~+NOT+)

- 不使用暫存器，交換兩個值 三種方法!! (XOR、加、乘)

$a \wedge 0 = a$

$a \wedge a = 0$

<https://blog.gtwang.org/programming/swap-two-numbers-without-third-temp-variable/>

- Big-Endian 與 Little-Endian

如果輸入是雙字組的 data 0x12345678，從記憶體位置 0x0000 開始寫入

地址	0x0000	0x0001	0x0002	0x0003
Big 值	0x12	0x34	0x56	0x78
Little 值	0x78	0x56	0x34	0x12

Big-Endian，最高位的位元組會放在最低的記憶體位址上

Little-Endian，最高位的位元組放在最高的記憶體位址上。

<https://blog.gtwang.org/programming/difference-between-big-endian-and-little-endian-implementation-in-c/>

- switch hub router gateway 差異

[http://bluemuta38.pixnet.net/blog/post/45543357-](http://bluemuta38.pixnet.net/blog/post/45543357-%E9%9B%86%E7%B7%9A%E5%99%A8%28hub%29%E3%80%81%E4%BA%A4%E6%8F%9B%E5%99%A8%28switch%29%E3%80%81ip%E5%88%86%E4%BA%AB%E5%99%A8%E3%80%81%E8%B7%AF%E7%94%B1%E5%99%A8)

[%E9%9B%86%E7%B7%9A%E5%99%A8%28hub%29%E3%80%81%E4%BA%A4%E6%8F%9B%E5%99%A8%28switch%29%E3%80%81ip%E5%88%86%E4%BA%AB%E5%99%A8%E3%80%81%E8%B7%AF%E7%94%B1%E5%99%A8](http://bluemuta38.pixnet.net/blog/post/45543357-%E9%9B%86%E7%B7%9A%E5%99%A8%28hub%29%E3%80%81%E4%BA%A4%E6%8F%9B%E5%99%A8%28switch%29%E3%80%81ip%E5%88%86%E4%BA%AB%E5%99%A8%E3%80%81%E8%B7%AF%E7%94%B1%E5%99%A8)

- 三向交握 (three-way hand shake) 四向交握(four-way hand shake)

三向交握 建立連線用

Alice 要和 Bob 交換訊息，Alice 先傳一個序列號給 Bob 要求同步(SYN)，Bob 收到後回傳確認

訊息說收到了(ACK)，同時 Bob 再傳一個序列號給 Alice 要求同步(SYN)，Alice 收到後回傳確認訊息說收到了(ACK)。成功建立連線。

四向交握 斷線用

A 發送一個斷線訊息給 B

B 收到後回發一個 ok 訊息給 A

接下來 B 在發送一個斷線訊息給 A

A 收到後再回發一個 ok 訊息給 B (這時 B 就正式斷線)

A 大約在 4 分鐘過後才正式斷線 (這時 A 就正式斷線，資源才會釋放)

<https://notfalse.net/7/three-way-handshake>

<https://ithelp.ithome.com.tw/articles/10205476>

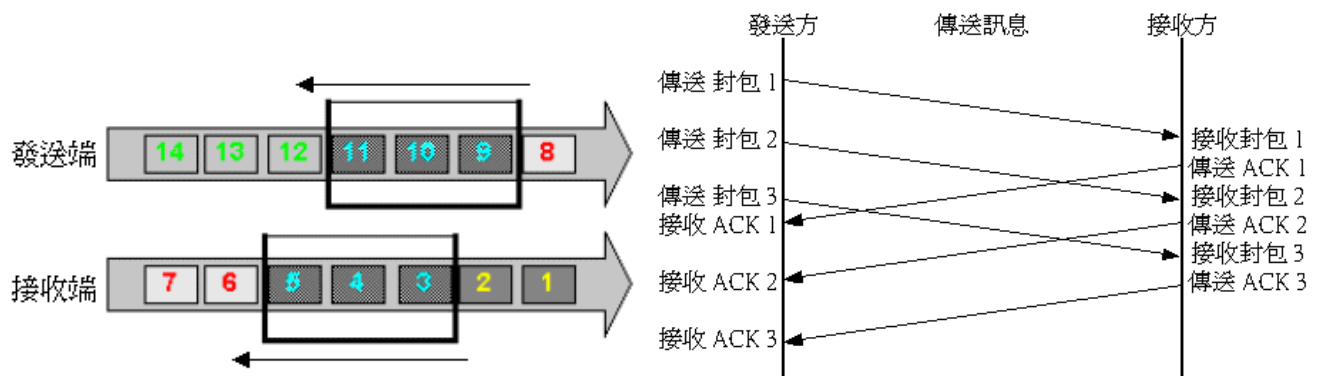
● TCP (Transmission Control Protocol) 連線導向(Connection Oriented)的可靠傳輸

接收端收到封包後，回傳 ACK 給傳送端，確認資料無誤後，雙方同時保留傳送封包的紀錄。

傳送端送出封包後，會有個專門的計時器，若接收端遲遲沒送 ACK 回來，會重新發送一次該封包，並重新計時；若還是沒收到 ACK，自動傳下一筆封包。

計時器雖然解決了封包丟失的問題，但如果封包的抵達只是因為網路延遲的關係沒有在預定時間完成，但卻在發送端重發後抵達，那麼，接收端就有可能接收到重複的封包。為解決這個困境，傳送協定會為每一個封包分配一個序號，並要求接收端按封包序號傳回確認信息。

假如每一個單一封包都需要需要等待前面的封包確認之後才進行傳送的話，這樣很浪費時間，所以發明「滑動視窗 sliding window」，多重發送和多重確認來加速。



我們可以根據 IP 來區別主機、根據埠口(port)來區別程式。一個 Socket 就是由一個 IP 與一個 Port 來定義的，Socket Pair 包含 4 個東西：

來源位址(Source Address)

來源埠口(Source Port)

目的位址(Destination Address)

目的埠口(Destination Port)

發送端	步驟	接收端
起始第一個封包 seq=1234567 data_len=100	1	
	2	收到第一個封包，將 100 bytes 的資料放進接收緩衝區內。
	3	發出第一個確認封包 ack=1234667 data_len=0 ACK_bit="on"
收到第一個確認，ack 值與 seq+data_len 比對正確。從發送緩衝區內減除 100 bytes 的資料。	4	
發送第二個封包 seq=1234667 data_len=150	5	
	6	收到第二封包，將 150 bytes 的資料放進接收緩衝區內。
	7	發出第二個確認封包，同時傳送第一筆回傳資料 seq=7654321 ack=1234817 data_len=50 ACK_bit="on"
收到第二個確認，ack 值與 seq+data_len 比對正確。從發送緩衝區內減除 150 bytes 的資料；同時將接收端送來的 50 bytes 資料放入接收緩衝區內。	8	
發送第三個封包，且回應確認剛才接收的資料 seq=1234817 ack=7654321 data_len=200 ACK_bit="on"	9	
	10	收到第三個封包以及第一個確認，ack 值與 seq+data_len 比對正確。從發送緩衝區內減除 50 bytes 的資料；同時將接收端送來的 200 bytes 資料放入接收緩衝區內。
.....	...	
.....	...	

適用收發 mail、檔案下載、網頁瀏覽

http://www.pcnet.idv.tw/pcnet/network/network_ip_tcp.htm

- UDP(User Datagram Protocol) 非連線型(Connectionless)的非可靠傳輸協定
不會運用確認機制來保證資料是否正確的被接收、不需要重傳遺失的資料、資料的接收可不按順序進行、也不提供回傳機制來控制資料流的速度。
適用某些訊息量較大、時效性大於可靠性的傳輸來說(比方說語音、影像、多媒體)。

● TCP UDP 比較

	優點	缺點
TCP	傳送可靠，程式可省略可靠機制	速度比較慢
UDP	傳輸量大，迅速	不可靠，程式或需自行提供可靠機制

- DHCP (Dynamic Host Configuration Protocol)
<https://zh.wikipedia.org/wiki/%E5%8A%A8%E6%80%81%E4%B8%BB%E6%9C%BA%E8%AE%BE%E7%BD%AE%E5%8D%8F%E8%AE%AE>

<https://docs.microsoft.com/zh-tw/windows-server/networking/technologies/dhcp/dhcp-top>

● 網路模型架構 (7 層 & 4 層)

7 層：實體層□資料連接層 □網路層□傳送層□會談層□表現層□應用層

4 層：網路存取層(結合前 2)□網路層□傳送層□應用層(結合後 3)

下層無需理會上層如何進行封裝，直接加自己的封裝，再把整個加封後的封包傳給更下一層。

實體層：應用在網路傳輸中的各種設備規格，以及如何將硬體所攜載的信號轉換成電腦可以理解的電子信號(0、1)，這通常都是設備上面之韌體(Firmware)的功能。

資料連接層：指定了要採用的信息單元(message unit，通常在 LAN 上面的信息單元被稱為 frame，翻譯為“訊框”或“框包”)，還有它們的格式、以及如何通過網路。每一個 frame 都

會被賦予一個 MAC 位址碼和偵錯監測值(checksum)。也會管網路卡的實體位址(Physical Address)，也被稱為 MAC(Media Access Control) Address。

網路層：介於網絡功能和使用功能之間。定義封包在網路中移動的路由和處理過程，讓封包(packet)在不同的網路之間成功地進行傳遞。如果封包不是屬於同一個網路時，會將之交由 router 處理。

傳送層：何控制節點之間的資料傳遞，還有錯誤檢測和修正的方法。

會談層：定義如何連接和掛斷連接。

表現層：定義數據的語法(syntax)、變更、和格式。將傳入的資料種類轉換成 PC 的資料種類，語法和格式都不同時，需轉換。

應用層：轉換應用程式相關的檔案格式。

https://www.pcnet.idv.tw/pcnet/network/network_ip_model.htm

- Linux 優缺點

<http://neuron.csie.ntust.edu.tw/homework/94/os/homework/homework1/A9415045/Linux%E7%89%B9%E8%89%B2%E4%BB%8B%E7%B4%B9.htm>

- Link list

<https://kopu.chat/2017/06/02/c-%E8%AA%9E%E8%A8%80%E7%B5%90%E4%B8%B2%E5%88%97linked-list%E7%9A%84%E5%BB%BA%E7%AB%8B%E8%88%87%E5%88%AA%E9%99%A4/>

[https://lakesd6531.pixnet.net/blog/post/329288496-c%E8%AA%9E%E8%A8%80%E7%B5%90%E4%B8%B2%E5%88%97\(link-list\)%E7%9A%84%E5%AF%A6%E4%BD%9C%E7%AF%84%E4%BE%8B](https://lakesd6531.pixnet.net/blog/post/329288496-c%E8%AA%9E%E8%A8%80%E7%B5%90%E4%B8%B2%E5%88%97(link-list)%E7%9A%84%E5%AF%A6%E4%BD%9C%E7%AF%84%E4%BE%8B)

- 資料型別

型別	符號位元	大小(byte)	表示方法	數值範圍
整數	有(用%d)	4	int 整數	-2147483648 ~ 2147483647
		1	char 字元	-128 ~ 127
		2	short (用%hd)	-32768 ~ 32767
		4	long (用%ld)	-2147483648 ~ 2147483647
		8	long long (用%lld)	
	無(用%u)	4	unsigned int (用%u)	0 ~ 4294967295
		1	unsigned char	0 ~ 256
		2	unsigned short (用%hu)	0 ~ 65535
		4	unsigned long (用%lu)	0 ~ 4294967295
		8	unsigned long long (用%llu)	
浮點數	有(用%f)	4	float 浮點數	$10^{-38} \sim 10^{38}$ 誤差與數值之比約為 10^{-7}

		8	double	$10^{-308} \sim 10^{308}$ 誤差與數值之比約為 10^{-16}
字元	有(用%c)	1	char	-128 ~ 127

輸出入格式符號	輸出入資料型態
%d	整數
%f	浮點數
%c	字元
%s	自串
%e	科學符號
%u	不帶符號 10 進位整數
%o	8 進位整數
%x	16 進位整數(小寫)
%p	位址(大寫)

int s = "abc"; printf("%s",s); “”用%s

int s = 'a'; printf("%c",s); ‘’用%c

char s = 'a'; 用%c 印出 a，用%d 印出 97(a 對應的 ASCII 碼)，用%s 錯誤

char s = 'abc'; 用%c 印出 c

char s = '456'; 用%c 印出 6

char s = "abc"; char 只能存單一字元，且必須用‘’，不能用”“

char s = "a"; 只能用‘’，不能用”“

int s[] = {3,87,21}; printf("%d",s[0]); 用陣列表示多個數字不用”“，直接{3,87,21}，記得用%d

int s[] = {'3','875','21'}; printf("%c",s[1]); 因為是%c，只會印最後一個字元，即 5

char s[] = {3,87,127}; printf("%d",s[2]); char 一樣可以印數字，只是範圍-127~127，其餘會錯

char s[] = {'3','875','21'}; printf("%c",s[1]); 因為是%c，只會印最後一個字元，即 5

和 int 相同，只能配‘’，不能配”“

int s[] = {'a','b'}; printf("%c",s[0]); printf("%c",s[1]); ‘’ 可以配上[]變多個(陣列)

”“不能配上[]變多個(陣列)

char s[] = {'abc','de','qaz'}; printf("%c",s[1]); 因為是%c，只會印最後一個字元，即 e

char s[] = "abc"; printf("%s",s); 唯一正確用陣列存單一文字 or 數字

單純 int s = 可以存文字(用”“配上%s)、單一字元(用‘’配上%c)、單純數字(%d)

單純 char s = 只能存單一字元或數字(用‘’配上%c)，

若存文字'abc'或'456'也只能%c 印出最後的字元 c 和 6 (不能%s)

完全不可用”“，只能唯一用‘’

以”陣列”方式存多個數字，可用 int 和 char(注意數字範圍，char 只能-127~+127)

{3,87,21} %d 該數字

{'3','875','21'} %c 最後一個字元

{"3","875","21"} 直接錯

以"陣列"方式存字元(只能用'存單一字元，或用'存很多個單一字元)(若用'存多個文字，也只能%c 出最後一個字元)，可用 int 和 char(但唯一能用')

{abc,de,qaz} 直接錯

{'a','b','c'} 只能'，不能"

{'abc','de','qaz'} 只能%c，且只印最後一個字元

{"abc","de","qaz"} 直接錯

以"陣列"方式存文字，只能存單一文字 or 數字(不可存多個文字)，唯一能用 char，且配上"

char s[] = "abc"; printf("%s",s); 唯一正確用陣列存單一文字 or 數字

char s[] = "456";

- printf 引數說明 %[旗標][寬度][.精度][長度修飾]資料型態
int a=10; %5d 印出__10(總共 5 個寬度，10 佔 2 個寬度，所以 10 左邊還有 3 個空白)
float a=-10.23456 %8.3f 印出_-10.234(總共 8 個寬度，正負、小數點都算 1 個寬度)
int a=10; %-5d 印出 10__(- 代表向左靠齊，總共 5 個寬度，右邊還有 3 個空白)
int a=10; %+5d 印出__+10(+ 代表強制印正負號)

%+10.3lld 印出 向左靠齊 有正負號 總共 10 個寬度 小數點取 3 位 的 long long int 值

- 字串

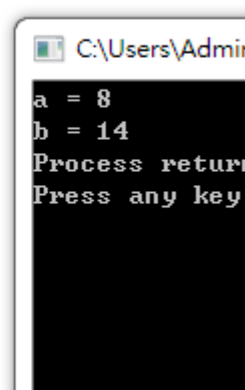
<https://programming.im.ncnu.edu.tw/Chapter11.htm>

- ++問題

```
int main()
{
    int a=6;
    int b=0;

    b = a++ + ++a;

    printf("a = %d\n",a);
    printf("b = %d",b);
}
```



a 原本是 6

a++代 6 進去，代完變 7

++a 代 8 進去

b = 6 + 8 = 14 此時 a 為 8

- lvalue rvalue 比較 (都是一個表達式(expression)而非物件(object))

lvalue 指明了一個續存物件(persistent object)或 function，例如++a,obj, *ptr, arr[index]

rvalue 是沒有名字的，可能是暫時物件(temporary object)，函數傳回的 non-reference value，或者就只是一個字面常數值，例如 a++,42,int(3.14)

lvalue 可以取址，所指的東西是有名字的、能續存下去的物件或函數，

rvalue 不能取址，所指的東西是沒有名字的、暫時性的、在表達式之後就會消失的東西。

- call by value 速度最慢，因為要複製一份再傳，傳的是值

	main的a	main的b		swap的c	swap的d
儲存值	5	10	5	10
記憶體位址	0x04	0x08	0x16	0x20

指標本身有佔記憶體位置

當副程式的程式執行完之後記憶體會變如下：

	main的a	main的b		swap的c	swap的d
儲存值	5	10	10	5
記憶體位址	0x04	0x08	0x16	0x20

call by address(即 call by value of pointer，value 本身是 address) 速度相同，原本的變數會被修改，傳的是址，可修改記憶體位置

```
1 // 宣告方式
2 void test(int *param) {
3     // param接收的是記憶體位址，所以呼叫時要特別注意
4     // 在function內對param做任何的異動都會影響到num
5     .....;
6
7     // 但是可以把param的記憶體位址再改掉
8     int temp;
9     param = &temp;
10 }
11
12 // 呼叫方式1
13 int num;
14 test(&num);
15
16 // 呼叫方式2
17 int *num;
18 test(num);
```

	main的a	main的b		swap的*c	swap的*d
儲存值	5	10	0x04	0x08
記憶體位址	0x04	0x08	0x16	0x20

指標本身有佔記憶體位置

所以副程式執行完交換後記憶體如下表示：

	main的a	main的b		swap的*c	swap的*d
儲存值	10	5	0x04	0x08
記憶體位址	0x04	0x08	0x16	0x20

call by reference 速度相同，原本的變數會被修改，傳的是值但用的是址，不能修改記憶體位置。傳參考就是變數的別名、綽號，因此副程式的記憶體位址跟主程式的位址一樣

	main的a	main的b		swap的&c	swap的&d
儲存值	5	10	5	10
記憶體位址	0x04	0x08	0x04	0x08

指標本身不佔記憶體位置，
跟原變數共用記憶體位置

<http://eeepage.info/call-by/>

<http://wp.mlab.tw/?p=176>

- FPGA (Field Programmable Gate Array. 電場可程式化邏輯閘陣列) 一種晶片，IC 驗證工具可重複程式設計的晶片，一般 IC 在設計完就不能改了，FPGA 是可重複設計。
在 IC Design 的初期，可以用 FPGA 驗證。
既解決了全客製化電路的不足，又克服了原有可程式化邏輯裝置閘電路數有限的缺點。
目前以硬體描述語言 (Verilog 或 VHDL) 描述的邏輯電路，可以利用邏輯合成和布局、布線工具軟體，快速地燒錄至 FPGA 上進行測試，這一過程是現代積體電路設計驗證的技術主流。
- ASIC (Application Specific Integrated Circuit 特殊應用積體電路)
依產品需求不同而客製化的特殊規格積體電路，成本極高，在確定投片前，用 FPGA 來實現 ASIC 原型，驗證他沒錯誤。
- ISR (interrupt service routine)
ISR 不能有返回值(不知道給誰)
ISR 不能傳遞參數(不知道誰呼叫)
不允許做浮點運算 (因為 ISR 應短而高效)
- Interrupt 中斷
處理器接收到來自硬體或軟體的訊號，提示發生了某個事件，應該被注意，這種情況就稱為中斷。
當處理器發出裝置請求後就可以立即返回以處理其他任務，而當裝置完成動作後，裝置傳送中斷訊號給處理器，處理器就可以再回過頭取得裝置處理結果。

- 設定一個絕對位址為0x67a9的整數型變數的值為0xaa55

Code

```
int *ptr;
ptr = (int *)0x67a9;
*ptr = 0xaa55;
```

- 判斷是否為2的倍數：把數字%10，取餘數，如果是0 2 4 6 8 就是偶數
判斷是否為3的倍數：x>0 就減3，x<0 就+3，最後x==0 即3的倍數
各別位數總和，再各別位數總和，直到為個位數x，x==3、6、9?

- 用Macro(巨集)求兩數最大值 `#define MAX(A,B) ((A)>=(B))?(A):(B)`

如何定義巨集?

語法 1: `#define` 巨集名稱 [替代內容]

語法 2: `#define` 巨集名稱(引數列) 運算式

功能：巨集名稱可用來定義常用的常數、字串、簡單的數學公式或函式。

(1)習慣大寫命名

(2)不用分號結尾

(3)當運算 overflow 時記得加上 UL

`#define SECONDS_PER_YEAR (60 * 60 * 24 * 365)UL`

(4)定義運算函式時要小心()的使用

- 字串轉浮點數 `atof` ex: `char s[]="3.678"; double f=atof(s);` f 輸出為 3.67800
字串轉整數 `atoi` ex: `char s[]="367"; int i=atoi(s);` i 輸出為 367

- 桶排序 Bucket sort (資料不能重複，且必須是正整數，且不能有0)
最穩定、最快、最浪費空間的排序法

準備10個空桶，最大數個空桶

[6 2 4 1 5 9] 待排數組

[0 0 0 0 0 0 0 0 0 0] 空桶

[0 1 2 3 4 5 6 7 8 9] 桶編號(實際不存在)

1, 順序從待排數組中取出數字, 首先6被取出, 然後把6入6號桶, 這個過程

[6 2 4 1 5 9]待排數組

[0 0 0 0 0 0 6 0 0 0]空桶

[0 1 2 3 4 5 6 7 8 9]桶編號(實際不存在)

2, 順序從待排數組中取出下一個數字, 此時2被取出, 將其放入2號桶, 是幾
[6 2 4 1 5 9] 待排數組
[0 0 2 0 0 0 6 0 0 0] 空桶
[0 1 2 3 4 5 6 7 8 9] 桶編號 (實際不存在)

3, 4, 5, 6 省略, 過程一樣, 全部入桶後變成下邊這樣

[6 2 4 1 5 9] 待排數組

[0 1 2 0 4 5 6 0 0 9] 空桶

[0 1 2 3 4 5 6 7 8 9] 桶編號 (實際不存在)

0 表示空桶, 跳過, 順序取出即可: 1 2 4 5 6 9

- 將字串 "I Love ILitek" 字串由小到大排列
大小比較直接用 $a[i] < a[j]$, 看有沒有成立就好, 程式在比字元大小時, 會自動轉 ASCII 做比較
- Linux 驅動程式解析
我們把外部的周邊裝置均視為一個檔案, 透過此檔案和實體硬體溝通, 那些檔案就稱 device file
Device file 的 major number 代表一個特定的裝置, ex: 硬碟有一個 major number
minor number 代表裝置上的子裝置,
ex: 同一個硬碟上的分割區就用不同 minor number 代表, 但其 major number 相同
- program process thread 比較
program: 放在二次儲存裝置中, 尚沒有被 Load 到記憶體的一堆 Code 稱之為「程式」。
process: 已經被 Load 到記憶體中, 任何一行 Code 隨時會被 CPU 執行, 且其宣告的在記憶體的變數的值會隨著需求而不斷變動。稱之為「程序」。(也就是活的 Program)
thread: 在同一個 Process 底下, 有許多自己的分身, 就是 Thread(執行緒), 同一個 Process 底下, 讓輸入文字是一個 Thread, 計算文字又是另外一個 Thread, 對 CPU 來說兩個都是類似一個 Process, 因此兩個可以同時做。每一個 Thread 之間可能會互搶資源, 而造成死結(Deadlock), 只要以下四個條件都滿足就有死結。
 - (1) 這個資源不能同時給兩個人用
 - (2) 有一個人拿了一個資源, 又想拿別人的資源
 - (3) 如果一個人占了茅坑不拉屎, 占用資源很久, 仍不能趕他走
 - (4) A 等 B, B 等 C, C 等 D, D 又等 A 等成一圈。
- mutex 和 semaphore 和 spinlock 常用的同步機制
mutex 透過一個變數或物件確保 Critical Section 內的資料同一時間內只會有單一存取。
semaphore 若 Critical Section 允許被兩個以上的執行緒執行 (但相對帶來的副作用用就是

Deadlock) (當 semaphore 一次只能被一個執行緒執行，又稱 binary semaphore 或 mutex)。

spinlock 一直等待指定的鎖被釋放之後，才可以繼續進行下一步動作。

mutex 只能被當初獲得的執行緒釋放，semaphore 則無此限制。

mutex 比較適合保護一塊區域(所需時間長)，而 spinlock 比較適合保護一個變數(所需時間短)。

- race condition(資源競爭) 類似 Deadlock

要阻止出現 race condition 情況的關鍵就是不能讓多個執行緒同時使用相同的 Critical Section。

- deadlock

一組 process 內的 thread 陷入互相等待的情況(Circular waiting)，造成 process 接無法往下執行，使得 CPU utilization 及 Throughput 大幅降低

Deadlock 成立的四個必要條件：互斥、持有並等待、不可強取豪奪、循環等待

- 遞迴 疊代 差別

遞迴法(recursive method)則是重複呼叫自身程式碼來得到答案，

疊代法(iterative method)是用迴圈去循環重複程式碼的某些部分來得到答案。

- pipeline 指令管線化 類似一個工廠多請幾個員工的概念

為了讓電腦和其它數位電子裝置能夠加速指令的通過速度（單位時間內被執行的指令數量）而設計的技術。

管線在處理器的內部被組織成層級，各個層級的管線能半獨立地單獨運作。

一般有 4 層管線的示意圖：讀取指令、指令解碼、執行指令、寫回執行結果。

未管線化的架構產生的效率低，因為有些 CPU 的模組在其他模組執行時是閒置的。

管線化雖並不會完全消除 CPU 閒置時間，但可讓這些模組並行運作而大幅提升程式執行效率。

如果一條指令管線能夠在每一個時脈週期接納一條新的指令，被稱為完整管線化（fully pipelined）。

缺點：某個指令突然錯誤，但下一個指令又在等他的值(兩指令有依賴關係)，會有衝突發生。

當一個指令從 Execute 階段到運算的資料能夠被其他指令使用，其中所經過的時間就叫 Instruction Latency。如果 pipeline 分太細，指令間又有依賴關係，處理器會空等。

Instr. No.	Pipeline Stage						
	IF	ID	EX	MEM	WB		
1							
2							
3							
4							
5							
Clock Cycle	1	2	3	4	5	6	7

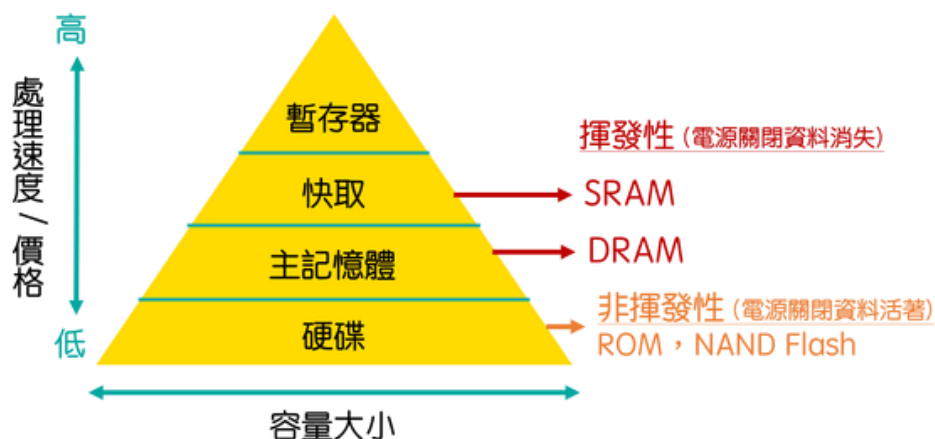
完成 1 個指令有 7 個動作，每個動作花 1 秒，以管線執行 100 個指令，需要多少時間？

第一個指令完成需要 7 秒，之後每 1 秒就多完成 1 個指令，所以需要 $7+(100-1)=106$ 秒

當指令數量很大的時候，用 pipeline 可以快 7 倍(幾個步驟)。

- overflow 打算存入的數值超過變數最大數值，就算 overflow 也不會有 error 訊息
underflow 處理的數值太小，就算 underflow 也不會有 error 訊息
- 記憶體 & 硬碟 RAM SRAM DRAM ROM
差異在於把電源關掉後、空間中儲存的資料還會不會留著。
「要被執行的程式和資料」保存在記憶體中是指電腦開啟後，從硬碟「複製一部份的資料」到記憶體裡面。

電腦中的儲存單元比較



CPU 是電腦的心臟，資料在此處理，程式指令也由它加以解釋，**暫存器**是 CPU 裡的儲存空間，在這運算完再存回去記憶體。

和 CPU 整合在一起的就是系統主記憶體，稱為**隨機存取記憶體(Random Access Memory, RAM)**，CPU 能夠不用按照位址的順序，而能隨機指定記憶體位址來讀取或寫入資料，可隨時更改記憶體內的資料非永久性質，RAM 又分成 DRAM、SRAM、VRAM

動態隨機存取記憶體 (DRAM) 慢(因為需重複充電): 用於電腦的主記憶體。資訊是存放在電容器裡的一系列電荷，但是電容器會漏電，所以需要不斷地充電以維持電位，所以稱為動態。

靜態隨機存取記憶體 (SRAM) 快(因為不用重複充電): 用於速度較快的快取記憶體。不必做自動充電的動作，只有寫入時會自動充電。

視訊隨機存取記憶體 (VRAM): 加速螢幕的顯示速度，像顯卡裡的 RAM。

唯讀記憶體(ROM): Read-Only Memory，電腦在運作時僅能從中讀取資料，而無法寫入新的資料，且不會隨電源關閉而消失。Ex:BIOS

Flash (快閃記憶體) 輕、小、功率低，分成 NOR Flash 和 NAND Flash。

NOR Flash 讀取速度較快，但寫入速度慢，用來儲存作業系統的程式碼或重要資料，ex:ROM。

NAND Flash 寫入的速度快、價格較低，較普遍。ex:USB、手機儲存空間、SSD。

→DRAM 便宜、主記憶體

揮發性→RAM →SRAM 貴、快取記憶體

非揮發性(硬碟)→ROM 只能讀，不能寫、BIOS

→Flash →NOR Flash 讀快寫慢、ROM

→NAND Flash 便宜、寫快、USB、SSD、手機內存

- C C++ C# 差別

C 最常應用在作業系統和韌體的開發及維護，保留字少，具有高階語言流程控制與資料處理的便利性，以及低階語言直接（包含以位元）操作記憶體的精密性。

C++ 是常應用於電腦軟體的開發及維護，物件導向特性、引入 reference。

C# 則是應用於網頁撰寫及架設居多，全物件導向設計的高階語言。

- driver 驅動程式 一種軟體

一種將硬體與作業系統相互連接的軟體。

- 虛擬記憶體 (Virtual Memory) 提升記憶體的使用度

讓應用程式認為其擁有連續可用的記憶體(一個連續完整的位址空間)，實際上，其通常是被分隔成多個實體記憶體碎片，還有部分暫時儲存在外部磁碟記憶體上，需要時才進行資料交換。

允許程式大小 > 實體記憶體 (physical memory)大小

提升記憶體的使用度

部分 program 在記憶體中執行即可，其餘的部分可以給其他 program 執行，提高 CPU 的效率

<https://mropengate.blogspot.com/2015/01/operating-system-ch9-virtual-memory.html>

- 第 N 個 bit 變成 1，先把 1 左移(<<)，和原數 OR(|) $((x) |= (1 << (N)))$

第 N 個 bit 變成 0，先把 1 左移(<<)，再 NOT(~)，和原數 AND(&) $((x) \&= (\sim(1 << (N))))$

第 N 個 bit 是 0 or 1，先把 1 左移(<<)，和原數 AND(&)，輸出即該 bit 為多少 $((x) \& (1 << (N))) != 0$

法 2: $((input) \& (1 << (N))) >> N$

第 N 個 bit 把 1 變 0，0 變 1，先把 1 左移(<<)，和原數 XOR(^) $((x) ^= (1 << (n)))$

把 1101 0010 第 3 個 bit 變成 1

把 0000 0001 左移 2($1 << 2$)，變成 0000 0100，和原本數字做 OR(|)

$0000\ 0100 \mid 1101\ 0010 = 1101\ 0110$

把 1101 0010 第 3 個 bit 變成 0

先把 0000 0001 左移 2($1 << 2$)，變成 0000 0100，再做 NOT(~)，變成 1111 1011，和原本數字做 AND(&)

$1111\ 1011 \& 1101\ 0010 = 1101\ 0010$

- Global 變數不給值，在 main 中印出來一律 = 0

- Global 變數 a=9 和 區域變數 a=10

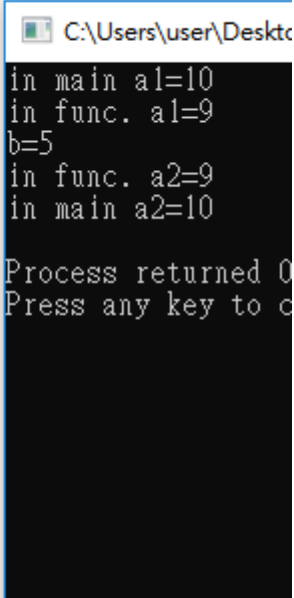
在 function 中求 a 是 global 的 a=9 (因為 function 根本沒宣告 a，所以 a 看 global 的值)

在 main 中求 a 是區域變數 a=10 (main 中有改 a 的值，所以用改過的值)

```
void out(int n);
int a=9;

int main()
{
    int a=10;
    int b=5;
    printf("in main a1=%d\n",a);
    out(5);
    printf("in main a2=%d\n",a);
}

void out(int n)
{
    printf("in func. a1=%d\n",a);
    printf("b=%d\n",n);
    printf("in func. a2=%d\n",a);
}
```



- #error: 強制 compiler 停止或中斷 compile 執行, 顯示錯誤訊息, 在**巨集處理階段就會停止**。
- #warning: 在編譯時, 輸出警告訊息, 警告使用者某些注意事項, 但是不會中止編譯, **仍然會繼續編譯**出目的檔。
- kernel 是作業系統的核心元件, 負責處理應用程式和硬體之間的溝通。
在 Linux 系統中 kernel 是主要組成部分。
- 作業系統(Operating System,OS)是管理電腦硬體與軟體資源的**系統軟體**, 同時也是電腦系統的核心與基石。
- struct → 所佔記憶體空間為 member 相加
union → 所佔記憶體空間由最大 size member 決定, 所以 union 的 member 同一時間只會最多出現一個
- int a[5]={10,11,12,13,14};
int *p=(int *)&a[1]; 一次跳一個 array 的大小(4*5=20 bytes)
*(a+1) = 11 一次跳一個 int 的大小
(*p+1) = garbage value
- 重新表示 void(*(*papf)[3])(char *);
typedef _____;
pf(*papf)[3];
答案是 typedef void(*pf) (char *);
扣除藍色一樣的部分, 再從第三行可以看到 function 名稱是 pf, 所以 typedef void(*pf) (char *);
因為他是 function pointer 的 typedef, new type name 是*後面的 pf
答案不是 typedef void(*) (char *) pf;

- `unsigned int zero = 0;`
`unsigned int compzero = 0xFFFF;` 錯誤的!!!!

應該用 `unsigned int compzero = ~0;` (才不會有處理器字長的問題)

- `#define dPS struct s *`
`typedef struct s * tPS;`

`dPS p1,p2;` `p1` 為一個指向結構的指，`p2` 為一個實際的結構
`tPS p3,p4;` `p3`、`p4` 為一個指向結構的指

- `int a = 5, b = 7, c;`
`c = a+++b;` 是 `c = a++ + b;` 從左向右讀，`++`優先權較高