**聯發科軟韌體筆記 WRITTEN BY 陳文遠**

1. **C語言**

* **const**

定義一個不允許被改變(read-only)的變數

|  |  |
| --- | --- |
| const int a[2] = {100, 200}; | 定義變數 a 為常數矩陣 |
| const int \*b = a; | a 為陣列首位址。b 為指向常數的指標。。 |
| int \*const c = a; | a 為陣列首位址。c 是常數指標。 |
| const int \*const d = a; | 指向常數的常數指針，指針本身與指向的內容都是不可改變的。 |
| const int func( ); | 表示函數返回值不可被修改。 |

* **判斷技巧：**如果 const 位於 \* 的左側，則 const 就是用來修飾指標所指向的變數，即指標指向為常量。如果 const 位於 \* 的右側，const 就是修飾指標本身，即指標本身是常量。
* **extern**

若有一個變數要在多個檔案中共用，就需要用 extern 來宣告。extern 會告訴 compiler 這個變數的存在。

例如你有以下三個檔案要 Link 執行：

|  |
| --- |
| **/\* example.c \*/**  #include <stdio.h>  #include "module.h"  int main(void){  printf("In main function, a = %d\n", a);  a = 0;  printf("In main function, a = %d\n", a);  hello();  return 0;  } |
| **/\* module.h \*/**  #ifndef MODULE\_H  #define MODULE\_H  **extern int a;**  void hello();  #endif |
| **/\* module.c \*/**  #include <stdio.h>  #include "module.h"  int a = 1;  void hello(){  printf("In hello function, a = %d\n", a);  } |

你會發現在 module.h 中宣告的 extern int a; 在 example.c 和 module.c 中是共用的，若修改 a 的值，則兩個檔案都會變。

* **static (for C only)**

static 在 C 中出現在不同的地方會有不同的意義，在 C 中有兩種可能：

1. **出現在變數前面，且該變數被宣告在某個 function 裡**

一般寫 function 時，在裡面宣告的變數會在 function 結束時被清除掉。但如果在變數宣告時加上 static，該變數就不會被清除。例如，你可以在函數中宣告 static int count; 來計算函數被呼叫了幾次。

|  |
| --- |
| void test(void){  static int count = 0;  printf("test function be called %dth times.\n", count++);  } |

1. **出現在變數前面，但該變數不是在某 function 裡**

當在 Link 編譯時會因為兩個檔案使用了相同的變數名而導致報錯，此時你可以在兩個檔案宣告變數前加上 static，告訴 compiler 此變數名僅在該檔案中有效。

|  |
| --- |
| **/\* module1.c \*/**  #include <stdio.h>  static int a = 1;  void print(void){  printf("a = %d\n", a);  } |
| **/\* module2.c \*/**  #include <stdio.h>  static int a = 2;  void print(void){  printf("a = %d\n", a);  } |

* **volatile**

被 volatile 宣告的變數將不會使用最佳化編譯，有時一個變數的值改變了 compiler 並不會馬上將他寫入記憶體中，而是先把結果放在CPU暫存器中，等到處理結束之後才寫入記憶體。

若這個變數是 multi-thread 的flag，且其他的thread需要透過這個變數來反應，而這個值卻又沒有寫入記憶體，這時便會取到錯誤的值。

又或者是這個變數為一個硬體的暫存器，且它會被硬體所改變，然而compiler 並沒有正確的將值從硬體暫存器取出來，而是將自己暫存的值拿來使用。這種情況，也需要用volatile 來宣告變數，以告訴compiler不要自己暫存變數來提升速度，如此一來這個變數只有任何的改變，便會馬上反應出來。

常見使用例子：

1. 硬體暫存器，s.t. 狀態暫存器
2. multi-thread 所共用的全域變數
3. 中斷服務函式 (Interrupt Service Rountine, ISR) 所使用的全域變數

* **Call by value (Pass by value)**

呼叫函數時，傳遞獨立的數值進去，當此數值在函數內被修改，則在函數外並不會被影響，以下面程式碼為例。

|  |
| --- |
| #include <stdio.h>  void change(int a){  a = 20;  }  int main(void){  int a = 10;  change(a);  printf("a = %d\n", a);  return 0;  } |

執行上方程式碼會發現輸出仍為 10，並不會因為函式內修改而改變。此方法速度也較慢，因為需要複製一份給給函式使用。

* **Call by reference (Pass by reference)**

呼叫函數時，傳遞到函式的參數與傳進去前的變數有連動，當在函式內修改其數值時，函數外的變數也會跟著被修改。

C 語言其實並沒有 Call by reference (C++才有)，但是你可以藉由傳遞指標來達成 Call by reference 的效果。以下為例子：

|  |
| --- |
| #include <stdio.h>  void setNum(int \*num2){  \*num2 = 5;  }  int main(void){  int num1 = 2;  printf("Before calling function, num1 = %d\n", num1);  setNum(&num1);  printf("After calling function, num1 = %d\n", num1);  return 0;  } |

你會發現經過 setNum( ) 函式後，num1 的值被修改成 5 了。

以下程式為使用 C++ 正確的 Call by reference 寫法

|  |
| --- |
| #include <iostream>  using namespace std;  void setNum(int &num2){  num2 = 5;  }  int main(void){  int num1 = 2;  cout << "Before calling function, num1 = " << num1 << endl;  setNum(num1);  cout << "After calling function, num1 = " << num1 << endl;  return 0;  } |

藉由在函式傳入值變數前面添加 & 來變成 reference type，來達到真正的 Call by reference 效果。

* **sizeof( ) 函式**

回傳系統中所占用的 bytes 數。

|  |  |  |
| --- | --- | --- |
|  | **32位元系統** | **64位元系統** |
| **sizeof(int)** | 4 | 4 |
| **sizeof(double)** | 8 | 8 |
| **sizeof(char)** | 1 | 1 |
| **sizeof(short)** | 2 | 2 |
| **sizeof(float)** | 4 | 4 |
| **sizeof(long)** | 4 | 8 |
| **sizeof(int\*)** | 4 | 8 |
| **sizeof(int\*\*)** | 4 | 8 |
| **sizeof(double\*)** | 4 | 8 |
| **sizeof(char\*)** | 4 | 8 |
| **sizeof(size\_t)** | 4 | 8 |
| **int mat[3] = {1, 2, 3}**  **sizeof(mat)** | 4\*3=12 | 4\*3=12 |

* **i++ 與 ++I 差別**

|  |
| --- |
| #include <stdio.h>  int main(void){  int i = 0;  int j = 0;  int A = i++; // 先將 i 放進 A 再加 1  int B = ++j; // 先將 j 加 1 再放進 B  printf("i = %d, j = %d\n", i, j);  printf("A = %d, B = %d\n", A, B);  return 0;  } |
| **輸出：**  i = 1, j = 1  A = 0, B = 1 |

* **使用 bit operation 來 swap 兩個變數**

|  |
| --- |
| #include <stdio.h>  int main(void){  int a = 10;  int b = 20;  a = a^b;  b = a^b;  a = a^b;  printf("a = %d, b = %d\n", a, b);  return 0;  } |
| **輸出：**  a = 20, b = 10 |

除了 bit operation 的方法外，也可以用加減法、乘除法來交換兩個變數。

1. **資料結構**

* **使用 malloc( ) 實作二維陣列 (5x5)**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  int main(void){  int \*\*a;  a = (int\*\*)malloc(5\*sizeof(int\*));  for(int i=0; i<5; i++)  a[i] = (int\*)malloc(5\*sizeof(int));  for(int i=0; i<5; i++)  for(int j=0; j<5; j++)  a[i][j] = i+j;  for(int i=0; i<5; i++){  for(int j=0; j<5; j++)  printf("%d ", a[i][j]);  printf("\n");  }  free(a);  return 0;  } |
| **輸出：**  0 1 2 3 4  1 2 3 4 5  2 3 4 5 6  3 4 5 6 7  4 5 6 7 8 |

* **struct 結構用法**

用於將一群具相關性的變數包裝成一個自行定義的型態。

|  |
| --- |
| #include <stdio.h>  #include <string.h>  struct Customer{  char name[10];  int age;  };  int main(void){  struct Customer c1, c2;  // 不可用 c1.name = "Chris";  memcpy(c1.name, "Chris", 10);  c1.age = 24;  memcpy(c2.name, "Rita", 10);  c2.age = 23;  printf("Customer 1 : name = %s, age = %d\n", c1.name, c1.age);  printf("Customer 2 : name = %s, age = %d\n", c2.name, c2.age);  return 0;  } |
| **輸出：**  Customer 1 : name = Chris, age = 24  Customer 2 : name = Peggy, age = 23 |

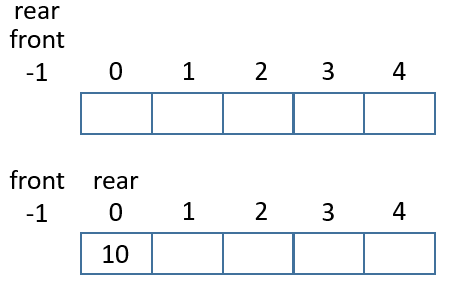
* **Linked List 鏈結串列**

實作使用 Linked List 串起數字 1~10。

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>  struct Number{  int num;  struct Number \*next;  };  int main(void){  struct Number \*head = (struct Number\*)malloc(sizeof(struct Number));  struct Number \*current = head;    for(int i=1; i<=10; i++){  current->num = i;  current->next = (struct Number\*)malloc(sizeof(struct Number));  current = current->next;  }  current = NULL;  current = head;  printf("Data in linked list: ");  while(current->next!=NULL){  printf("%d ", current->num);  current = current->next;  }  printf("\n");  return 0;  } |
| **輸出：**  Data in linked list: 1 2 3 4 5 6 7 8 9 10 |

* **Queue 佇列**

Queue 的操作模式是先進先出(First in first out, FIFO)，類似於排隊的概念，排在前面的會先被輸出。以下我用陣列來實作。



* **將資料放進 Queue 中 (enqueue)**

需要預先定義兩個變數 front=-1 以及 rear=-1，front 代表最前的資料的 index，rear 代表最後面資料的 index。

|  |
| --- |
| void enqueue(char num[10]){  if(rear>=MAX-1)  printf(KRED\_L"\n[ERROR] The Queue is full!\n");  else{  rear++;  strcpy(queue[rear], num);  printf(KGRN\_L"\n[SUCCESS] Insert OK!\n");  }  } |

* **將資料從 Queue 中刪除 (dequeue)**

原則就是越早放入的資料越先刪除，作法也很簡單，只需要將 front++ 即可

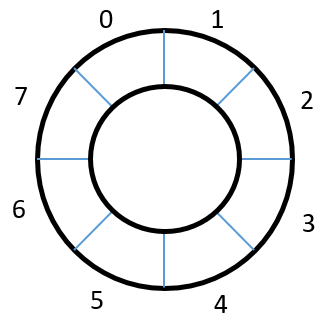
|  |
| --- |
| void dequeue(){  if(front==rear)  printf(KRED\_L"\n[ERROR] The Queue is empty.\n");  else{  front++;  printf(KGRN\_L"\n[SUCCESS] Delete ok!\n");  }  } |

* **列出 Queue 中所有資料 (list)**

|  |
| --- |
| void list(){  if(front==rear)  printf(KRED\_L"\n[ERROR] The Queue is empty.\n");  else{  printf(KGRN\_L"\nContent\n");  printf("-------------\n");  for(int i=front+1; i<=rear; i++)  printf("%s\n", queue[i]);  printf("-------------\n");  }  } |

以上的 Queue 例子中你會發現，rear 跟 front 變數都不斷增加，最後會導致 Queue 中明明還有位置，但卻無法放資料，此問題可以應用**環狀佇列**來解決。

* **Circle Queue 環狀佇列**



* **在環狀佇列中加入資料 (enqueue)**

|  |
| --- |
| void enqueue(char num[10]){  if((rear==MAX-1 && front==-1) || (rear+1)%MAX==front)  printf(KRED\_L"\n[ERROR] Circle Queue is full!\n");  else{  rear = (rear+1) % MAX;  strcpy(queue[rear], num);  printf(KGRN\_L"\n[SUCCESS] Insert ok!\n");  }  } |

* **從環狀佇列中拿出資料 (dequeue)**

同樣秉持著先進先出的原則。

|  |
| --- |
| void dequeue(){  if(front==rear)  printf(KRED\_L"\n[ERROR] Circle Queue is empty.\n");  else{  front = (front+1) % MAX;  printf(KGRN\_L"\n[SUCCESS] Delete ok!\n");  }  } |

* **列出環狀佇列中的所有資料 (list)**

越早存進來的資料要越早輸出。

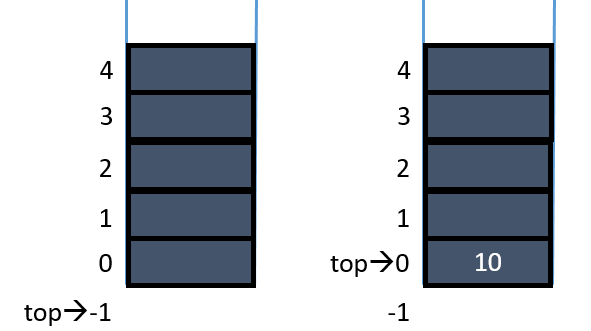
|  |
| --- |
| void list(){  if(front==rear)  printf(KRED\_L"\n[ERROR] Circle Qeueue is empty.\n");  else{  printf(KGRN\_L"\nITEM\n");  printf("------------\n");  for(int i=(front+1)%MAX; ; i=(i+1)%MAX){  printf("%s\n", queue[i]);  if(i==rear)  break;  }  printf("------------\n");  }  } |

* **完整環狀佇列程式碼**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #ifndef \_DEBUG\_COLOR\_  #define \_DEBUG\_COLOR\_  #define KDRK "\x1B[0;30m"  #define KGRY "\x1B[1;30m"  #define KRED "\x1B[0;31m"  #define KRED\_L "\x1B[1;31m"  #define KGRN "\x1B[0;32m"  #define KGRN\_L "\x1B[1;32m"  #define KYEL "\x1B[0;33m"  #define KYEL\_L "\x1B[1;33m"  #define KBLU "\x1B[0;34m"  #define KBLU\_L "\x1B[1;34m"  #define KMAG "\x1B[0;35m"  #define KMAG\_L "\x1B[1;35m"  #define KCYN "\x1B[0;36m"  #define KCYN\_L "\x1B[1;36m"  #define WHITE "\x1B[0;37m"  #define WHITE\_L "\x1B[1;37m"  #define RESET "\x1B[0m"  #endif  #define MAX 3  int front = -1;  int rear = -1;  char queue[MAX][10];  void enqueue(char num[10]){  if((rear==MAX-1 && front==-1) || (rear+1)%MAX==front)  printf(KRED\_L"\n[ERROR] Circle Queue is full!\n");  else{  rear = (rear+1) % MAX;  strcpy(queue[rear], num);  printf(KGRN\_L"\n[SUCCESS] Insert ok!\n");  }  }  void dequeue(){  if(front==rear)  printf(KRED\_L"\n[ERROR] Circle Queue is empty.\n");  else{  front = (front+1) % MAX;  printf(KGRN\_L"\n[SUCCESS] Delete ok!\n");  }  }  void list(){  if(front==rear)  printf(KRED\_L"\n[ERROR] Circle Qeueue is empty.\n");  else{  printf(KGRN\_L"\nITEM\n");  printf("------------\n");  for(int i=(front+1)%MAX; ; i=(i+1)%MAX){  printf("%s\n", queue[i]);  if(i==rear)  break;  }  printf("------------\n");  }  }  int main(void){  int option;  char num[10];  while(1){  puts(KBLU\_L"");  puts("|----------------------------------------|");  puts("|We provide the following functionality. |");  puts("|----------------------------------------|");  puts("|[1] Insert (Enqueue) |");  puts("|[2] Delete (Dequeue) |");  puts("|[3] List all data |");  puts("|[4] Quit |");  puts("|----------------------------------------|");  printf(KYEL"\nEnter your choice: ");  option = getchar();  getchar();  switch(option){  case '1':  printf(KCYN"\nEnter the item(int) you want to add: ");  fgets(num, sizeof num, stdin);  strtok(num, "\n");  enqueue(num);  break;  case '2':  dequeue();  break;  case '3':  list();  break;  case '4':  printf(KGRN\_L"\n[SUCCESS] Quit.\n\n");  exit(0);  default:  printf(KRED\_L"\n[ERROR] You enter the wrong command. Please try again.\n");  }  }  return 0;  } |

* **Stack 堆疊**

堆疊是一種有序串列 (order list)，其加入或刪除數據的動作都在同一端 (top 端)。加入元素的動作稱為 Push，而刪除元素的動作稱為 Pop。由於堆疊 (Stack) 有著先進去的元素最後才會被搬出來的特性，因此又可稱為 LIFO (Last In First Out) 串列。以下使用陣列實作堆疊。



* **將資料插入堆疊中 (push)**

需要先定義一個 top=-1 變數作為此堆疊陣列的 index，也必須先定義好堆疊陣列的大小(MAX)。

|  |
| --- |
| void push(char str[20]){  if(top>=MAX-1)  printf(KRED\_L"\n[ERROR] Stack is full!\n");  else{  top++;  strcpy(item[top], str);  printf(KGRN\_L"\n[SUCCESS] Insert OK!\n");  }  } |

* **將資料從堆疊中刪除 (pop)**

每次刪除資料都只會刪除堆疊的最上層，也就是最後一個放進的資料，這是 stack 的特性。刪除資料很簡單，只需要把 top 這個 index 往下移一個就行。

|  |
| --- |
| void pop(){  if(top<0)  printf(KRED\_L"\n[ERROR] No data in stack.\n");  else{  top--;  printf(KGRN\_L"\n[SUCCESS] Pop the top item.\n");  }  } |

* **將堆疊中全部資料列出 (list)**

將堆疊陣列中的資料列出，越晚放進的要越早輸出。

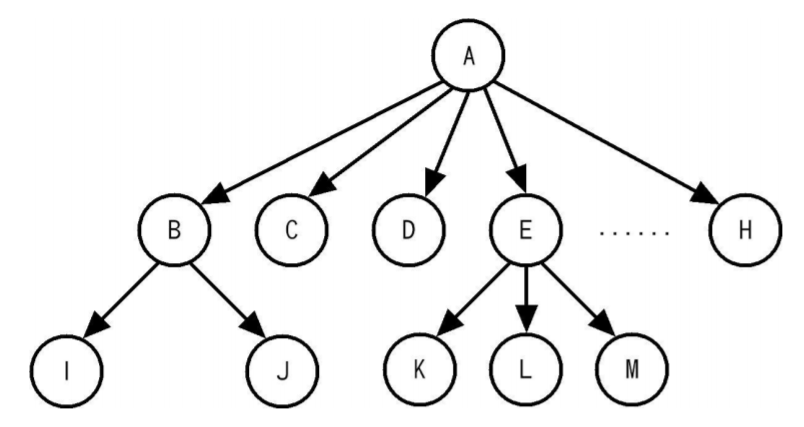
|  |
| --- |
| void list(){  printf(KGRN\_L"\nITEM\n");  printf("------------\n");  for(int i=top; i>-1; i--)  printf("%s\n", item[i]);  printf("------------\n");  } |

* **堆疊的完整程式碼**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #ifndef \_DEBUG\_COLOR\_  #define \_DEBUG\_COLOR\_  #define KDRK "\x1B[0;30m"  #define KGRY "\x1B[1;30m"  #define KRED "\x1B[0;31m"  #define KRED\_L "\x1B[1;31m"  #define KGRN "\x1B[0;32m"  #define KGRN\_L "\x1B[1;32m"  #define KYEL "\x1B[0;33m"  #define KYEL\_L "\x1B[1;33m"  #define KBLU "\x1B[0;34m"  #define KBLU\_L "\x1B[1;34m"  #define KMAG "\x1B[0;35m"  #define KMAG\_L "\x1B[1;35m"  #define KCYN "\x1B[0;36m"  #define KCYN\_L "\x1B[1;36m"  #define WHITE "\x1B[0;37m"  #define WHITE\_L "\x1B[1;37m"  #define RESET "\x1B[0m"  #endif  #define MAX 10  int top = -1;  char item[MAX][20];  void push(char str[20]){  if(top>=MAX-1)  printf(KRED\_L"\n[ERROR] Stack is full!\n");  else{  top++;  strcpy(item[top], str);  printf(KGRN\_L"\n[SUCCESS] Insert OK!\n");  }  }  void pop(){  if(top<0)  printf(KRED\_L"\n[ERROR] No data in stack.\n");  else{  top--;  printf(KGRN\_L"\n[SUCCESS] Pop the top item.\n");  }  }  void list(){  printf(KGRN\_L"\nITEM\n");  printf("------------\n");  for(int i=top; i>-1; i--)  printf("%s\n", item[i]);  printf("------------\n");  }  int main(void){  int option;  char str[20];  while(1){  puts(KBLU\_L"");  puts("|----------------------------------------|");  puts("|We provide the following functionality. |");  puts("|----------------------------------------|");  puts("|[1] Insert (Push) |");  puts("|[2] Delete (Pop) |");  puts("|[3] List all data |");  puts("|[4] Quit |");  puts("|----------------------------------------|");  printf(KYEL"\nEnter your choice: ");  option = getchar();  getchar();  switch((char)option){  case '1':  printf(KCYN"\nEnter the item(string) you want to add: ");  fgets(str, sizeof str, stdin);  strtok(str, "\n");  push(str);  break;  case '2':  pop();  break;  case '3':  list();  break;  case '4':  printf(KGRN\_L"\n[SUCCESS] Quit.\n\n");  exit(0);  default:  printf(KRED\_L"\n[ERROR] You enter the wrong command. Please try again.\n");  }  }  return 0;  } |

* **Tree 樹**





1. **演算法**

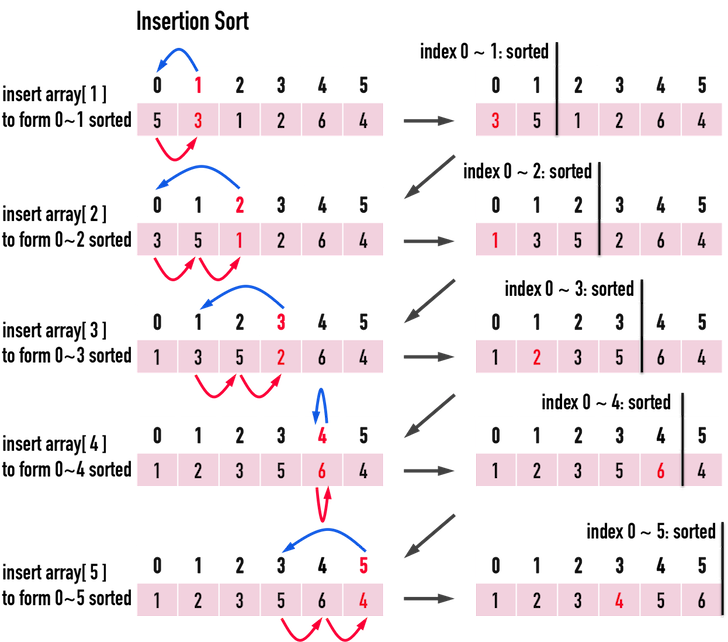
* **Bubble Sort 泡沫排序法**

給定一組數列，對這個數列進行巡訪，當目前這個數字大於後面數字時，就將這兩個數字交換，如此一來，當這次巡訪完之後，最大的數字就會移到最後一個。接著，再從頭開始重新巡訪此數列，重複剛剛的動作，第二輪巡訪完之後，就會將第二大的數字移到倒數第二個，不斷重複此動作，直到第一個數字就是我們的移動目標位置，這樣整個數列就由小到大排序完成了。

|  |
| --- |
| #include <stdio.h>  #include <stdbool.h>  #define LEN 8  void print\_array(int num[]){  for(int i=0; i<LEN; i++){  printf("%d ", num[i]);  }  printf("\n");  }  void bubble\_sort(int num[]){  bool has\_sorted;  for(int i=LEN-1; i>=0; i--){  has\_sorted = false;  for(int j=0; j<i; j++){  if(num[j] > num[j+1]){  num[j] = num[j] ^ num[j+1];  num[j+1] = num[j] ^ num[j+1];  num[j] = num[j] ^ num[j+1];  }  has\_sorted = true;  }  if(has\_sorted==false)  break;  }  }  int main(void){  int num[LEN] = {2, 12, 7, 5, 43, 20, 9, 37};  printf("Before sorting: ");  print\_array(num);  printf("After sorting: ");  bubble\_sort(num);  print\_array(num);  return 0;  } |
| **輸出：**  Before sorting: 2 12 7 5 43 20 9 37  After sorting: 2 5 7 9 12 20 37 43 |

* **Insertion Sort 插入排序法**

直接看下圖比較好理解。

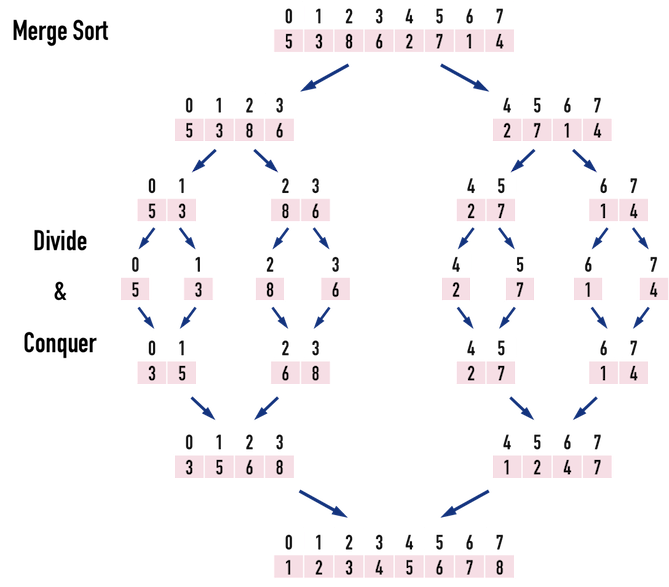


接著看 insertion sort 程式碼

|  |
| --- |
| void insertion\_sort(int num[]){  for(int i=1; i<LEN; i++){  for(int j=0; j<i; j++){  if(num[i] < num[j]){  num[i] = num[i] ^ num[j];  num[j] = num[i] ^ num[j];  num[i] = num[i] ^ num[j];  }  }  }  }  // 其他部分跟上面的 bubble sort 一樣 |

* **Merge Sort 合併排序法**

將數列做 divide 對半拆解，最後做 conquer 將其合併成有序數列，如下圖。



* Selection Sort 選擇排序法
* Quick Sort 快速排序法
* Binary-Search Tree 二元搜尋樹

1. 作業系統
2. 計算機結構(組織)
3. 常見的 Protocol