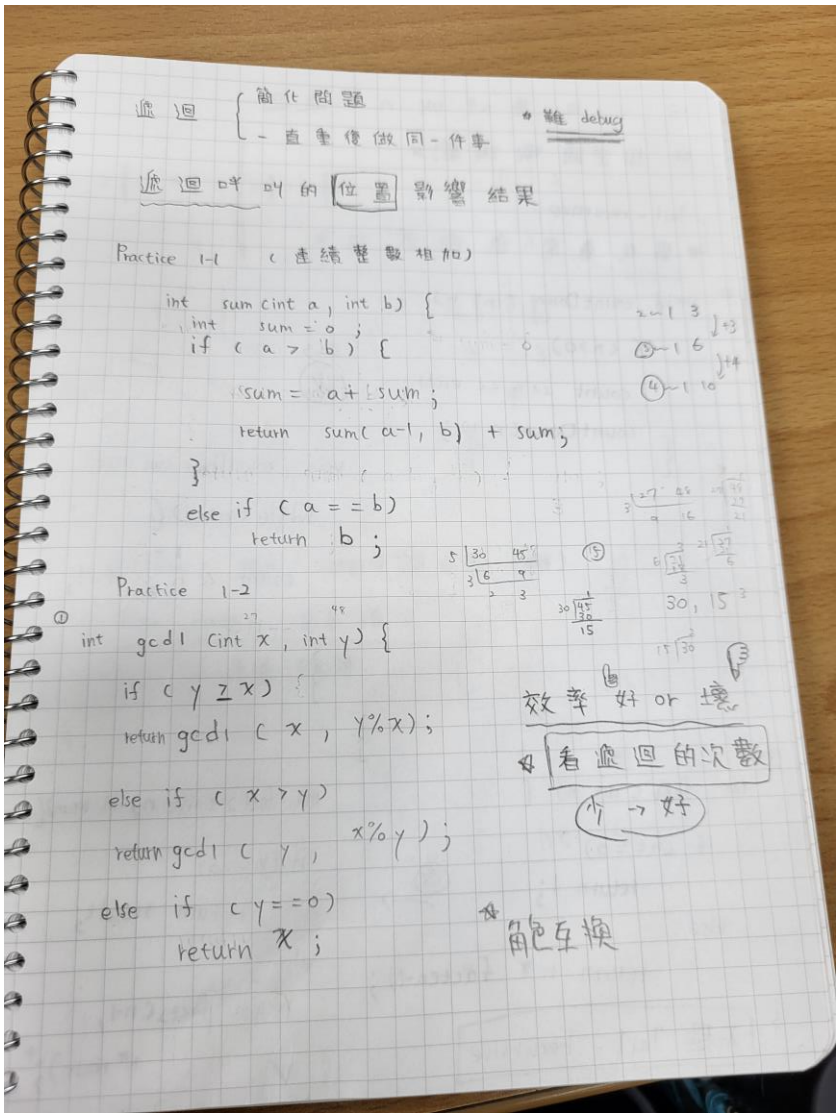


遞迴 { 簡化問題
- 直重複做同一件事

* 難 debug

遞迴呼叫的位置影響結果

Practice 1-1 (連續整數相加)

```
int sum(int a, int b) {
    int sum = 0;
    if (a > b) {
        sum = a + sum;
        return sum(a-1, b) + sum;
    }
    else if (a == b)
        return b;
}
```

Practice 1-2

```
int gcd1(int x, int y) {
    if (y >= x)
        return gcd1(x, y%x);
    else if (x > y)
        return gcd1(y, x%y);
    else if (y == 0)
        return x;
}
```

效率好 or 壞

* 看遞迴的次數

少 → 好

* 能互換

n元 → 每一層呼叫 n 次遞迴

★ 用空間換時間 ★

Tail - recursive

↳ 最後會是遞迴式結束

①

```
void countDown (int n) {
    if (n>0) {
        count << n << endl;
        countDown(n-1);
    }
}
```

改 →

```
void countDown (int n){
    while (n>0){
        count << n <<endl;
        --n;
    }
}
```

②

```
int fact (int n) {
    if (n==0)
        return 1;
    else
        return n * fact(n-1);
}
```

不是 Tail - recursive

改 →

```
int fact2 (int n, int result){
    if (n==0)
        return result;
    else
        return fact2(n-1, n* result);
}
```

✓

物件導向

- data members => 記錄基本資料
- methods => 功能，動作

Encapsulation 封裝 => hides inner details

Inheritance 繼承 => reused (超連結)

Polymorphism 多型 => 執行

Operation Contracts 運算合約 (分析問題)

- 目的 (purpose)
- 假設 (Assumptions)
- 輸入 (Input)
- 輸出 (Output)

* 薪水高的時候，要跟人事說
提撥的比例要提高 *

Cohesion 高內聚 - modules perform single well-defined tasks

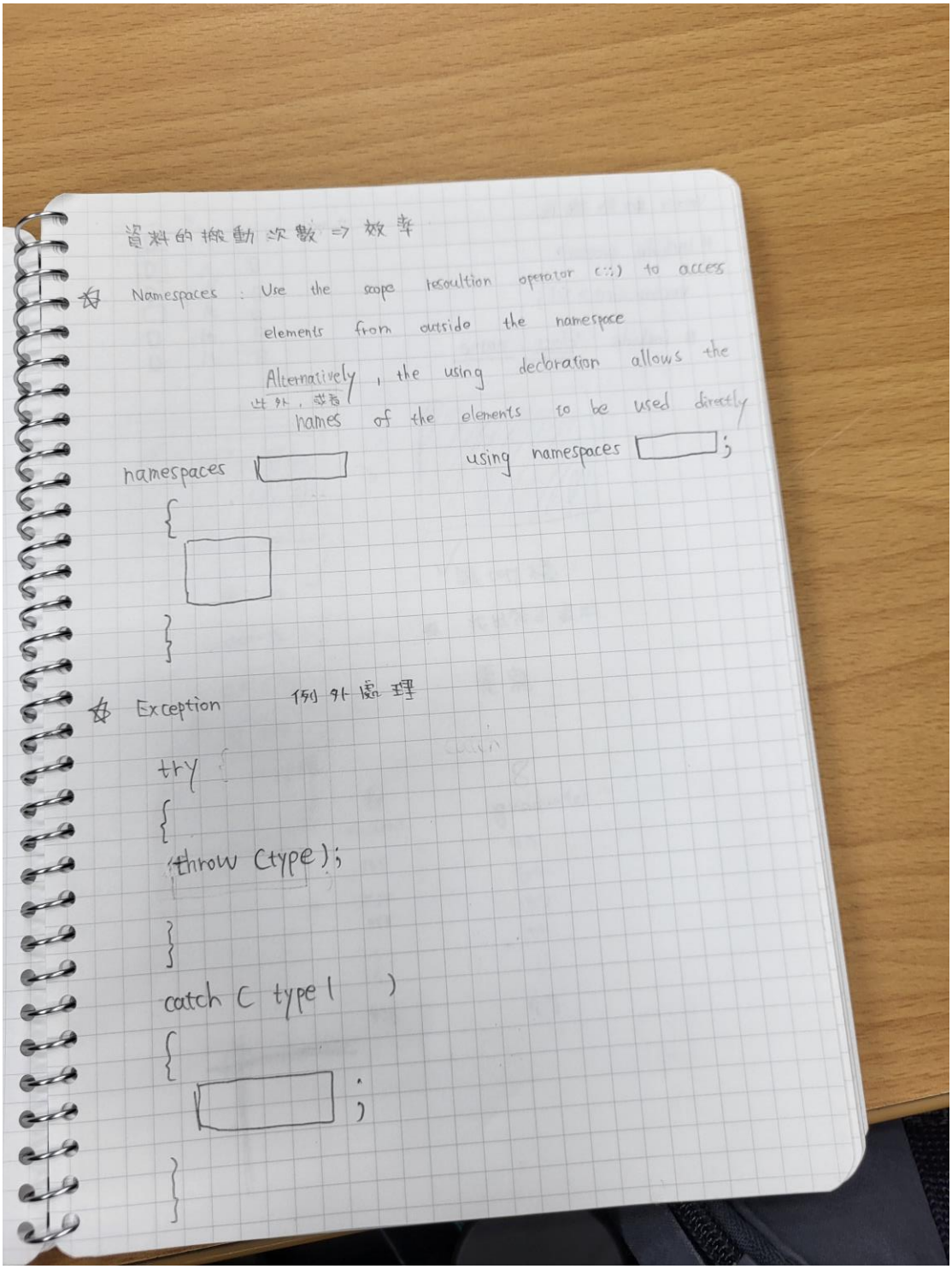
Coupling 低耦合 - measure of dependence among modules

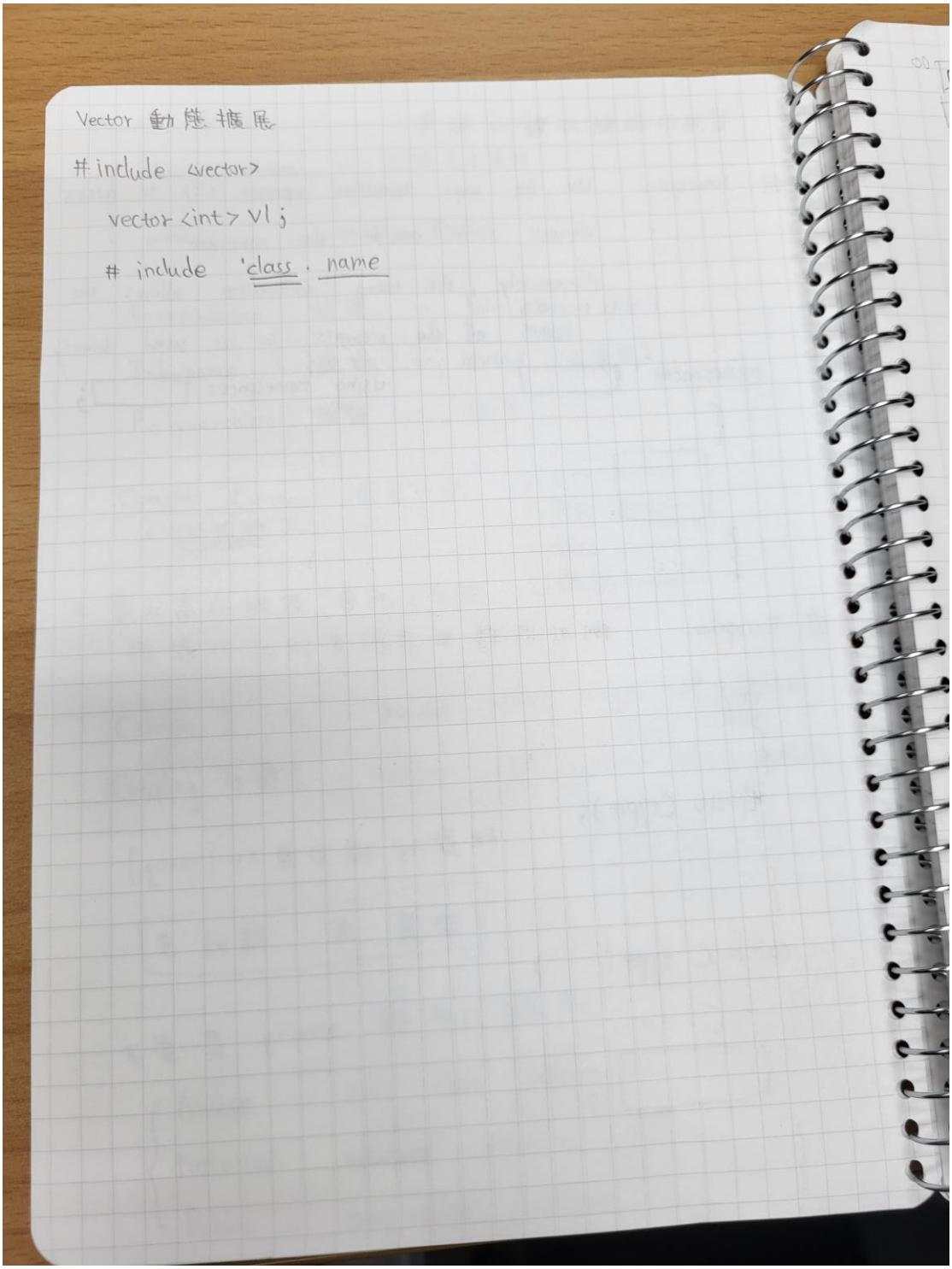
{query} => 不會改動到資料

先設計 再實作

不要是 private 都可以繼承

- Private : only class instances
- Protected : subclass instances
- Public : any class instances


資料的搬動次數 => 效率
Namespaces : Use the scope resoultion operator (::) to access elements from outside the namespace
Alternatively, the using declaration allows the 此外，或者
names of the elements to be used directly
namespaces
using namespaces ;
Exception 例外處理
try
{
throw Ctype);
}
catch C type1)
{
;
}

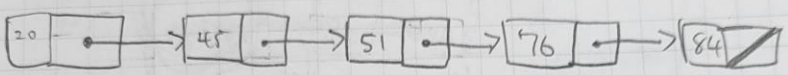

Vector 動態擴展
#include <vector>
vector<int> v1;
#include 'class . name

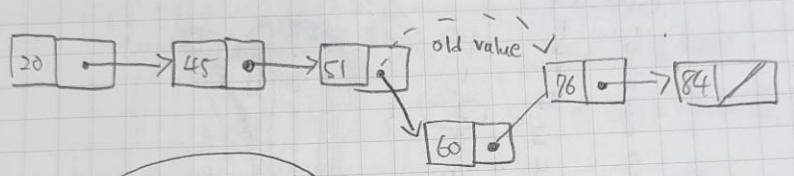
Linked Lists

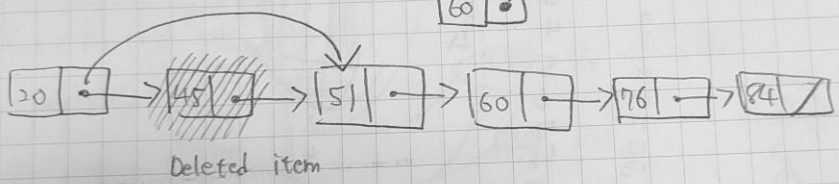
排頭很重要

記憶體

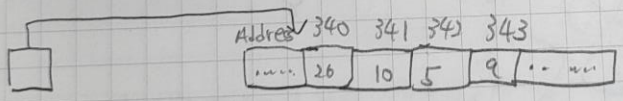
指標







指標 = 門牌



delet 完 設 NULL

動態配置陣列

```
int arraySize = 50;
double *anArray = new double[arraySize];
anArray[2] = *(anArray+2)

double *oldArray = anArray;
anArray = new double[3*arraySize]
```

重新

```
for(int index=0; index<arraysize; index++)
    anArray[index] = oldArray[index];

delete [] oldArray;
```

一群

```
struct Node {
    int item;
    Node *next;
};
Node *p
p=new Node
```

刪除：

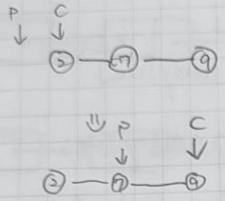
```
□->next = NULL;
delete □;
□ = NULL;
```

order→[/|]

找到正確位置

Node *prev *cur;

for (prev = NULL , cur = head ; (cur != NULL) &&
(newValue > cur→ item) ; (prev = cur, cur = cur
→next);



停住

陣列 省空間

鏈結串列 : 耗空間

輸出檔案

ofstream

for (Nod

outFile

outFile .

ifstream

讀檔

ifstream

int

if

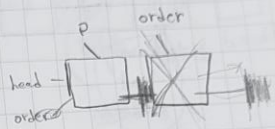
輸出檔案

ofstream outFile (fileName)

for (Node *cur = head; cur != NULL; cur = cur->next)

outfile << cur->item <<endl

outFile.close();



ifstream

讀檔

ifstream inFile (fileName);

int nextItem;

if (inFile >> nextItem) {

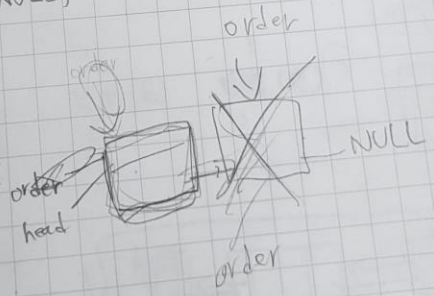
head = new Node

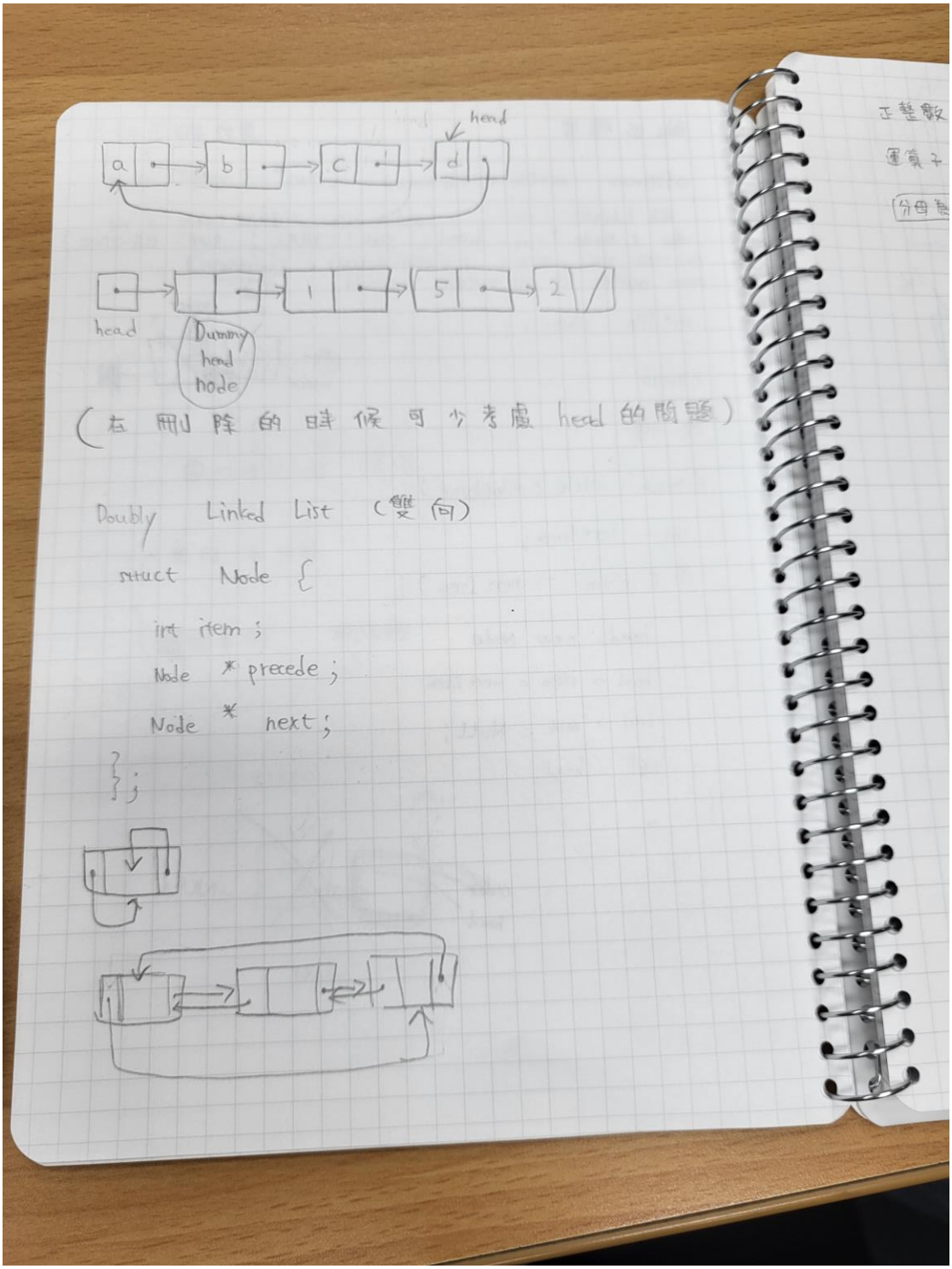
head -> item = nextItem

head -> next = NULL;

tail = head;

}




head
a
b
c
d
head
1
5
2
Dummy head node
(在刪除的時候可少考慮head的問題)
Doubly Linked List (雙向)
struct Node {
int item;
Node *precede;
Node *next;
};
正整數
運算子
分母

Defining Languages

x | y x 或 y

- xy or x • y means x followed by y

<words>

<identifier> = <letter> (base) | <identifier> <letter> |
<identifier> <digit>

(遞迴)

ex: A2C

③ A | ① A2 C |

② A 2

```
is Id (in : w:string): boolean
  if (w is of length 1)
    if (w is a letter)
        return true;
    else
        return false
  else if (the last character of w is a letter
                                or a digit)
      return isId (w minus its last character)
  else
    return false.
```

回文

<pal> = empty string <ch> (base) | a <pal> a |

b <pal> b | ……

<ch> = a | b | c ……

A^nB^n

<legal-word> = empty string |

A <pal> B

Infix expression 中序 a+b ☆(先後順序)

Prefix expression 前序 +ab

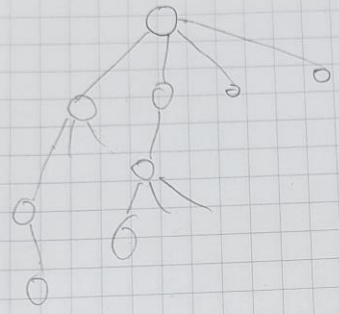
Postfix expression 後序 ab+

不需定義誰要先做

☆ 一個前序式後面再接上非空字串一定不是前序式

Back tracking

與遞迴有關



不行就退回上一個

數學歸納法 => 確保遞迴是否正確

```
if ( n is 0 )
    return 1
else
    return  n * fact (n-1)
```

Basis (Base case)

Inductive hypothesis

.

Inductive step