遞迴

原理:兩面鏡子無限延伸,鏡子愈来愈小

效果:把問題縮小,程式精簡,易解釋

Ex: Factorial (階乘), Greatest Common Divisor (最大公因數)
Search in Array (搜尋), Fibonacci series 費式數列
Combinatorial number 組合數 Tower of Hanoi 河内塔, fractal

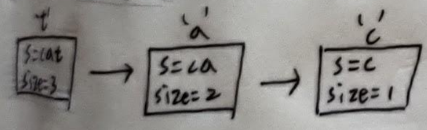
binary search:

divide and conquer 分而擊之

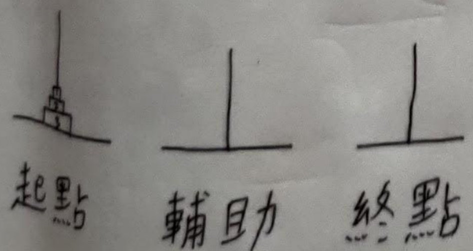
倒過来印:

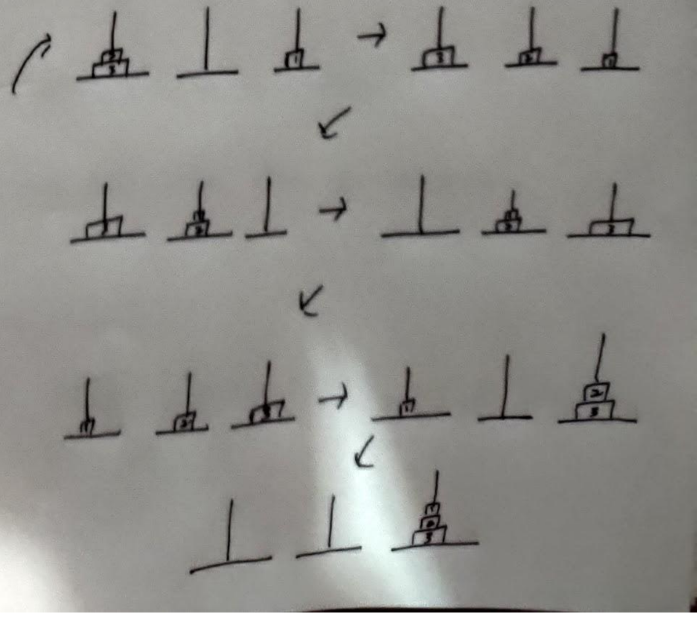
把字串長度減一,減少字元:問題縮小

長度等於零:中止條件



Tower of Hanoi





3盤: 2^3-1

64盤: $2^{64}-1$

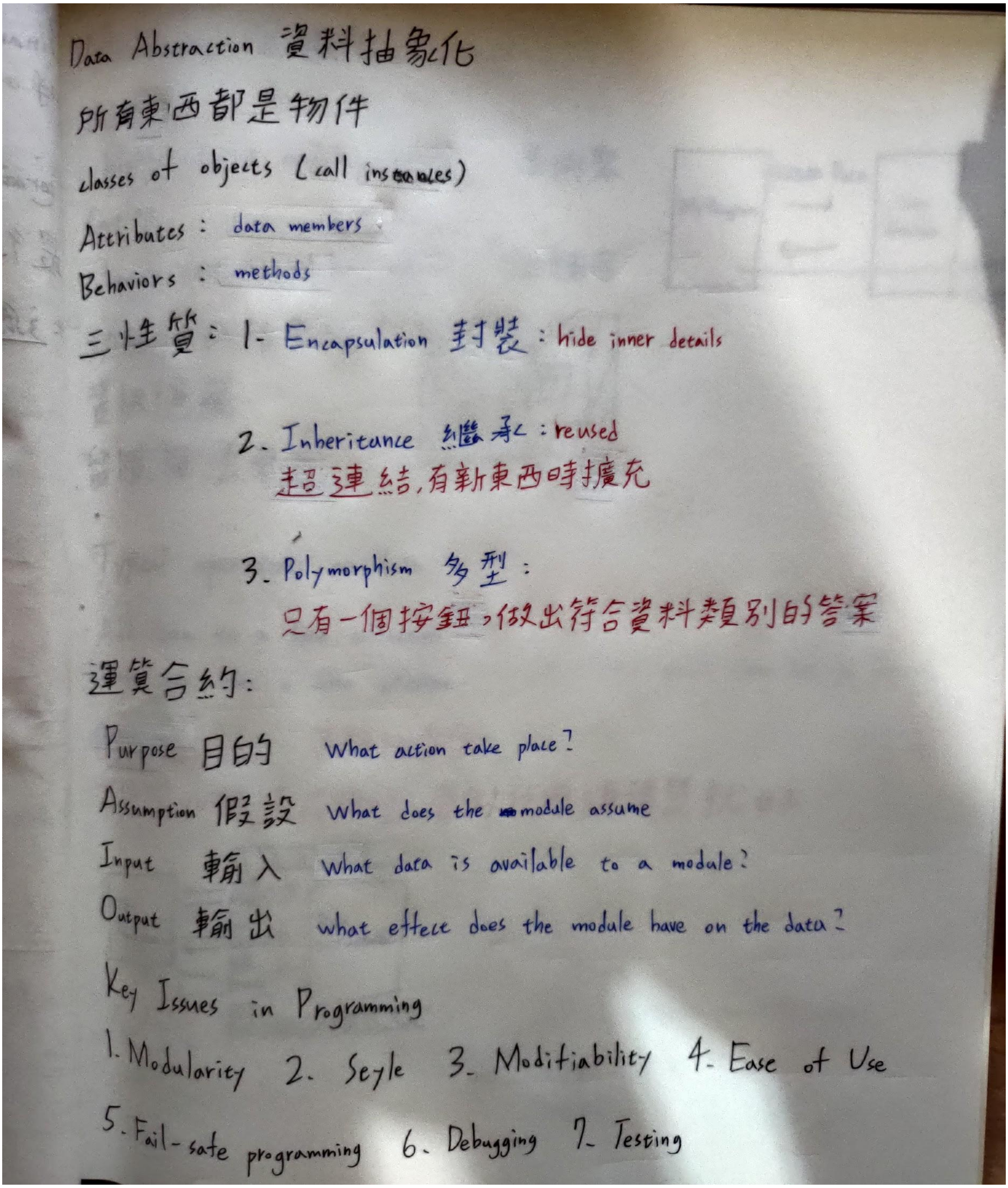
Binary Recursion (二元遞迴)

呼叫兩次遞迴

尾端遞迴:

最後一個是遞迴呼叫,可被轉成迴圈

*遞迴不一定有效率,只是精簡易看



Data Abstraction 資料抽象化

所有東西都是物件

classes of objects (call instances)

Attributes : data members

Behaviors : methods

三性質 : 1. Encapsulation 封裝 : hide inner details

2. Inheritance 繼承 : reused
超連結，有新東西時擴充

3. Polymorphism 多型 :
只有一個按鈕，做出符合資料類別的答案

運算合約 :

Purpose 目的 What action take place?

Assumption 假設 What does the module assume

Input 輸入 What data is available to a module?

Output 輸出 what effect does the module have on the data?

Key Issues in Programming

1. Modularity 2. Seyle 3. Modifiability 4. Ease of Use

5. Fail-safe programming 6. Debugging 7. Testing

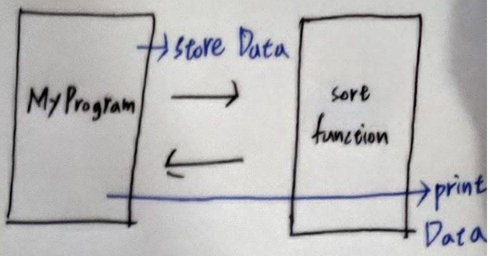
Modularity 模組化

Cohesion

- highly cohesive modules desired 高内聚

Coupling

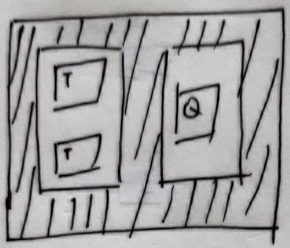
- Loosely coupled modules desired 低耦合



功能性的抽象化

資訊隱藏

留個洞,丟東西

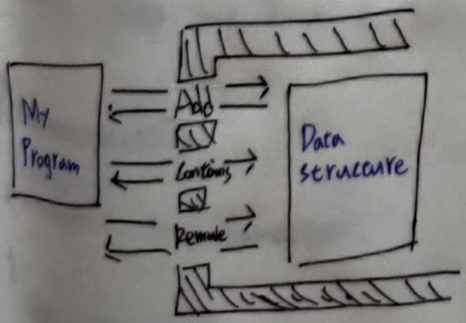


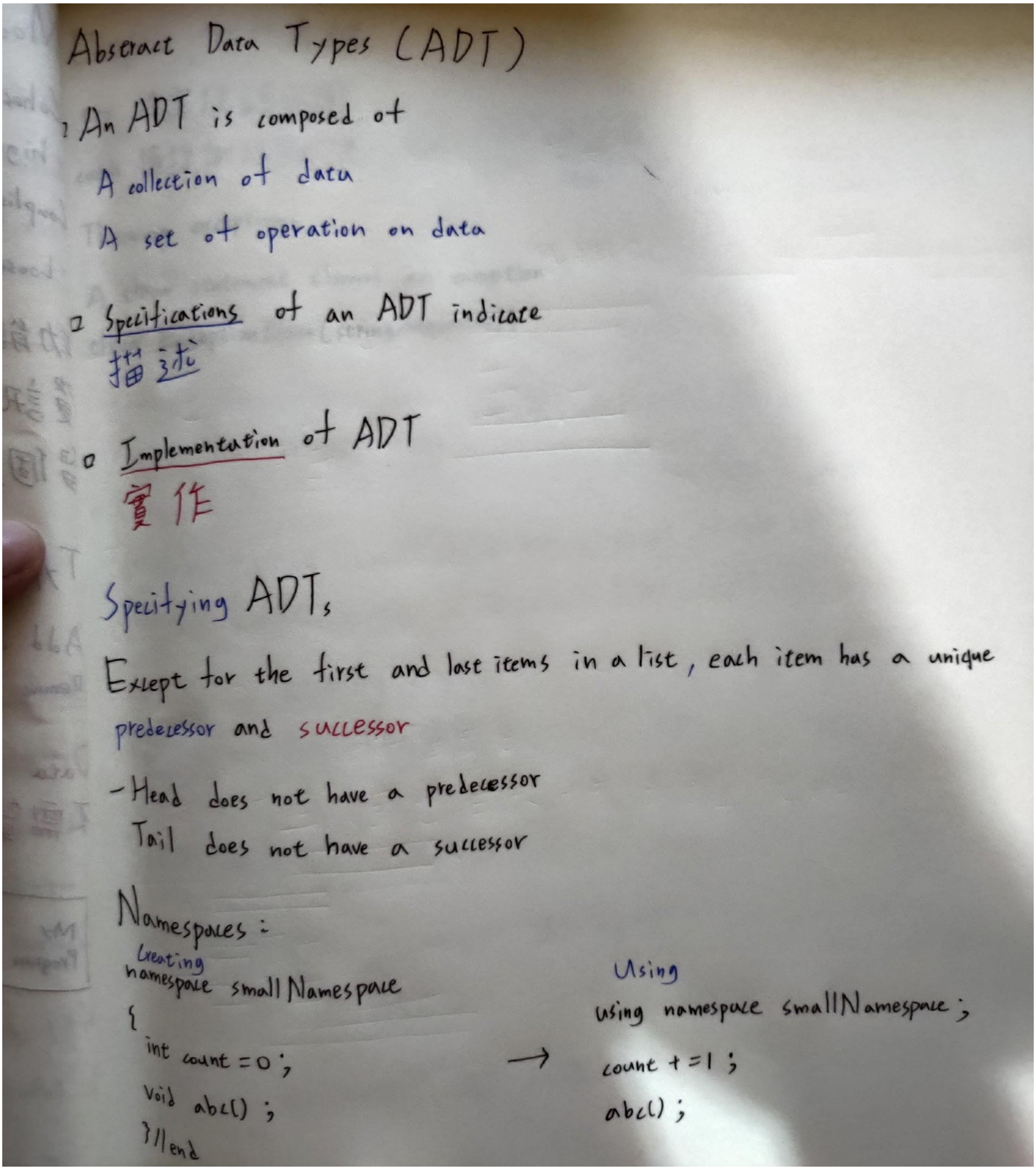
Typical operations on data

- Add data to a data collection
- Remove data from a data collection

Data abstraction 資料抽象化

不需管如何達成目地,要做什麼講清楚就好





Abstract Data Types (ADT)

- An ADT is composed of

  A collection of data

  A set of operation on data

□ Specifications of an ADT indicate

描述

□ Implementation of ADT

實作

Specifying ADTs

Except for the first and last items in a list, each item has a unique predecessor and successor

- Head does not have a predecessor

  Tail does not have a successor

Namespaces:

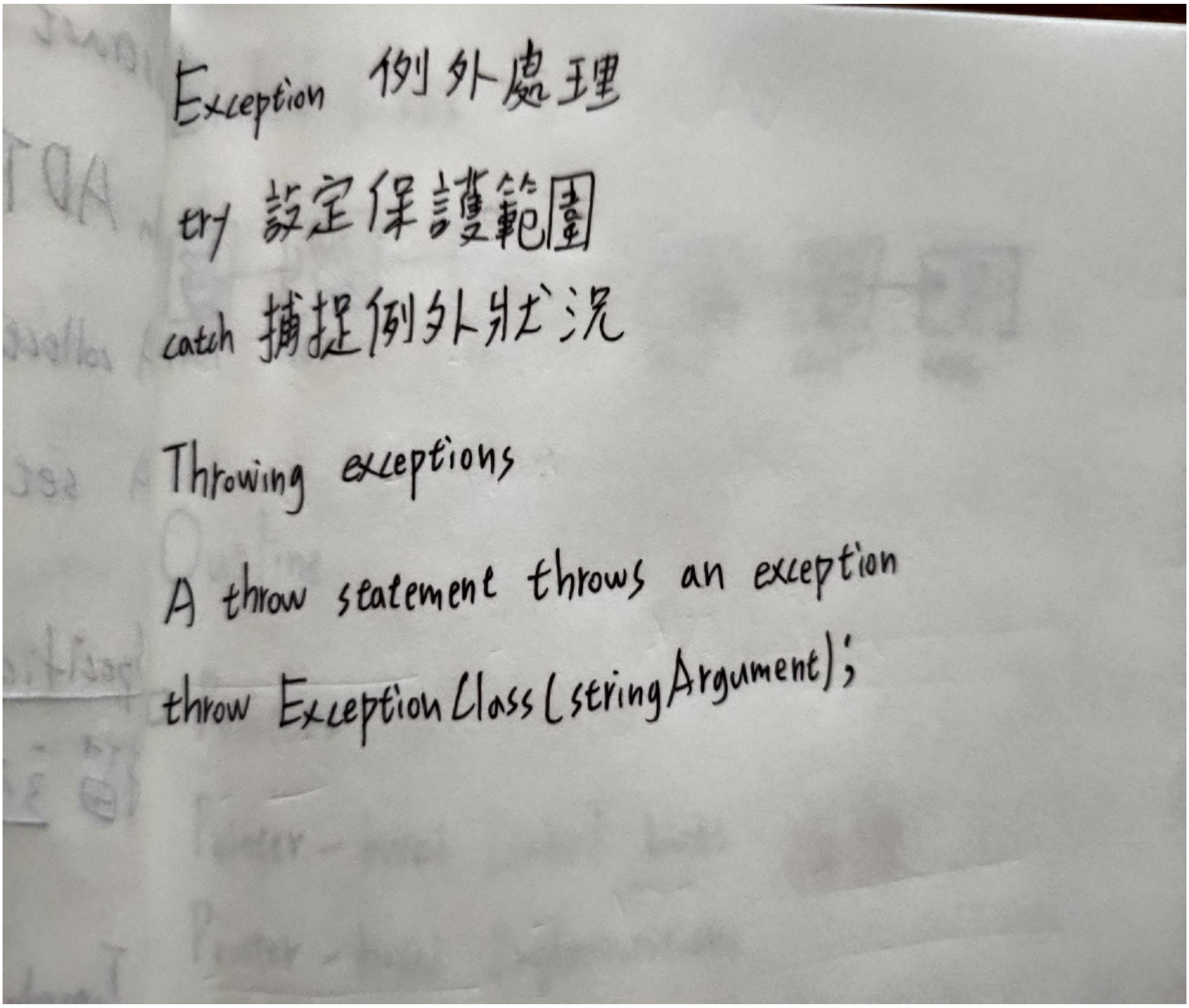
Creating

```
namespace smallNamespace
{
  int count = 0;
  void abc();
}//end
```

→

Using

```
using namespace smallNamespace;
count += 1;
abc();
```



Exception 例外處理

try 設定保護範圍

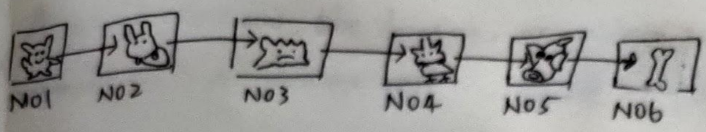
catch 捕捉例外狀況

Throwing exceptions

A throw statement throws an exception

throw ExceptionClass(stringArgument);

Linked Lists :



Outline

1 Pointers

Pointer - based Linked Lists 指標

Pointer - based Implementations

2 Variations

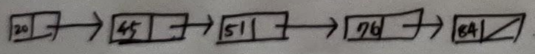
Circular Linked Lists 環狀鏈結串列

Doubly Linked Lists 雙向鏈結串列

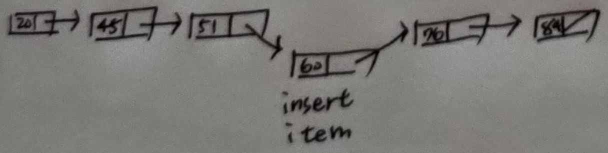
陣列 vs 鏈結串列

需移動資料　不需移動資料

(a)

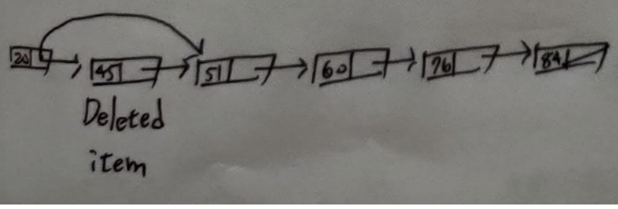


(b)



新增

(c)



刪除

Pointers

A pointeer contains the location, or address in memory, of a memory cell 指標=門牌

- Declaration of an integer variable P

int *P

Intially undefined, but not NULL 還沒有房子

Static allocation 一般變數:直接配給

The expresion *P represent the memory cell to which p points

To place the address of a variable into a pointer variable, you can use

The address of operator &

p = &x;

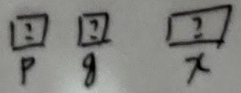
The new operator &x=房子x的門牌

p= new int;

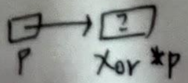
delete p; 歸還房子

p=NULL 徹底遺忘門牌!

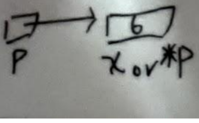
(a) 申請空白門牌: int *p, *q;
int x;



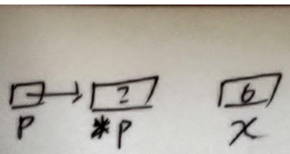
(b) 抄寫別人的門牌: p=&x;



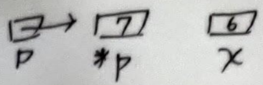
(c) 鳩佔鵲巢~ : *p=6;



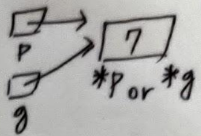
(d) 緊急配置：p = new int;



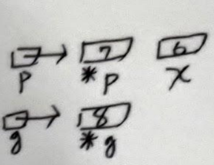
(e) 堆放家當：*p = 7;



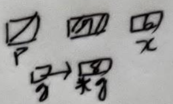
(f) 抄寫至另一張門牌：q = p;



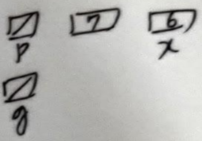
(g) 緊急配置並堆放家當：g = new int
*g = 8



(h) 遺忘門牌 ： p=NULL



(i) 歸還房子並遺忘門牌：delete g;
g = NULL



Dynamic Allocation of Arrays : 動態陣列

You can use the new operator to allocate an array dynamically

```
int arraySize = 50;
double *anArray = new double [arraySize];
```

資料要一個一個搬

```
delete [] oldArray;
```
歸還

```
double *oldArray = anArray;
anArray = new double [3*arraySize];
```
配置更大空間