資結筆記

單元1：遞迴

- 一、基本概念

  * 迴圈 v.s. 遞迴（寫一份程式碼就好）

    - 無限延伸
    - 漸小
    - 精簡

    ⇒ 問題愈來愈小直到不見

  Type

  Linear Recursion 線性遞迴

  Binary Recursion 二元遞迴

  應用

  Factorial 階乘 / Greatest Common Divisor 最大公因數

  Search in Array / Fibonacci Series 費氏數列

  Combinational numbers / Tower of Hanoi 河內塔

  Example: A binary search in recursive

  - Repeatedly halves the data collection and searches the one half that could contain the item
  - Uses a divide and conquer strategy (分而擊之)

- 一、以反向印出字串遞迴範例①

  Problem：字串長度減一

  Sol: diminishes by 1 the length of the string

  Base Case: write the empty string backward, minus last character.

```
void writeBackward ( string s, int size) {
    if (size > 0) {
        cout << s.substr (size-1, 1);    // 輸出
        writeBackward (s, size-1);       // 遞迴呼叫
    } //if
}
```

- —— 以反向印出字串遞迴範例②

```
void writeBackWard2 (string s, int size){
  if (s is empty)
  else {
     writeBackWard2 ( s minus its first chracter ); // 呼叫
     cout << s.substr() ;                           // 輸出
  } // else
}
```

- —— Greatest Common Divisor

① gcd1 (x,y) = x if y=0. → Base case
= gcd1 (x, y mod x) if y>x
= gcd1 (y, x mod y) else.

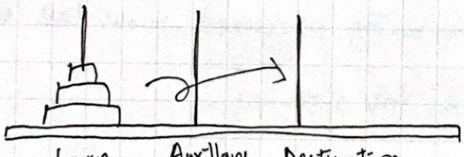
② gcd2 (x,y) = y if x mod y = 0
= gcd (y, x mod y) else.

① vs ②

① [x=9, y=6 gcd1(6,3)=?] → [x=6, y=3 gcd1(3,9)=?] → [x=3. y=3 return 3;]

② [x=9, y=6 gcd2(6,3)=?] → [x=6, y=3 return 3;]

- —— Towers of Hanoi



⇒ 個數 n ⇒ 公式: $2^n - 1$ #

```
solveTowers ( count, source, destination, spare)
if (count == 1) Move disk from source to destination ;
else {
     solveTowers(count-1, source, spare, destination);
     solveTowers (1, source, destination, spare) ;
     solve Towers ( count-1, spare, destination, source) ;
}
```

- Fibonacci 費氏數列

$n_k = n_{k-1} + n_{k-2}$ $(n_k \geq 2^{k/2})$

```
linearFibonacci(k) {
  if (k==1) return (k, 0);
  else
      (i, j) = linearFibonacci(k-1)
      return (i+j, i);
  }
}
```

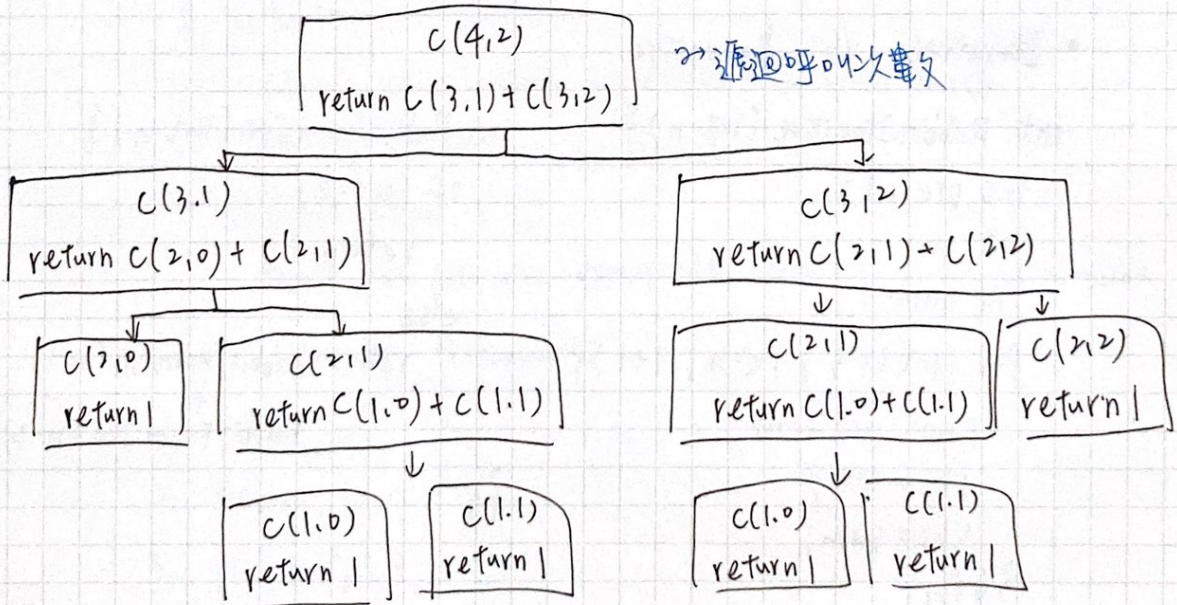
- Choose K out of N. things.

$C(n,k) = 1$ if $k=0$

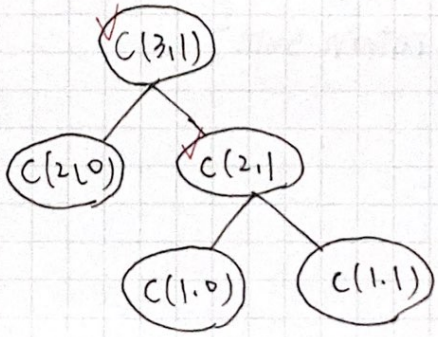
$= 1$ if $k=n$

$= 0$ if $k>n$

$= C(n-1,k-1) + C(n-1,k)$ if $0<k<n$



- Binary Tree = 元樹.
  - Leaf nodes 葉節點
  - Internal nodes 內部節點
  - (leaf nodes) − (internal nodes) = 1



- tail Recursion 尾端遞迴

  ⇒ 整個程式碼最後一個指令是遞迴

  ⇒ 易改成迴圈

EXAMPLE:

```
void CountDown(int n){
  if(n>0){
    cout << n << endl;
    CountDown(n-1);
  }// if
}// CountDown
```

⇒

```
while(n>0){
  cout << n << endl;
  n--;
}//while
```

- Four Question about Recursion
  1. 遞迴定義
  2. 問題簡化
  3. 終止條件
  4. 保證終止 base case

- Iterative vs Recursive

```
int RabbitIterative(int n){
  int pre=1;
  int cur=1;
  int sum=1;
  for(int i=3; i<=n; i++){
    sum=pre+cur;
    pre=cur;
    cur=sum;
  }//for
  return sum;
}//
```

```
int RabbitRecursive(int n){
  if(n<=2)
    return 1;
  else
    return RabbitRecursive(n-1)
           +
           RabbitRecursive(n-2);
}//
```

單元2：抽象化 Abstraction

- 物件導向 Object-Oriented Programming
  - Attributes of objects of a single type 屬性
    - Typically Data
    - Called data members
  - Behaviors (operations)
    - Typically operate on the data 運算
    - Called methods or member function

△ Principles of Object-Oriented Programming

1, Encapsulation 封裝
- object combine data and operation
- "Hides" inner details

2, Inheritance 繼承
- Classes can Inherit properties from other classes
- Existing classes can be "reused"

3, Polymorphism 多型
- Objects can determine appropriate operations at "execution" time

△ Key Issue in Programming

1, Modularity
2, Style
3, Modifiability
4, Ease of Use
5, Fail-safe programming
6, Debugging
7, Testing

- 資料抽象化 Data Abstraction

  △ motives 動机
  - Modularity 模組化
  - Cohesion 高內聚
  - Coupling 低耦合
  - Functional abstraction 功能性的抽象化

  △ concepts 概念 - the isolation of modules is not total

  △ goals 目標
  - Typical operations on data
  - data abstraction

  △ ADT
  - composed of
    - collection of data
    - set of operations on that data
  - Specifications of an ADT indicate 描述
    - what the ADT operations do, not how to implement that
  - Implementation of an ADT 實作
    - Includes choosing a particular data structure

- 反轉序列 Reverse List

  △ 反轉整個序列

(一) reverseList (in aList: List, out source: boolean)

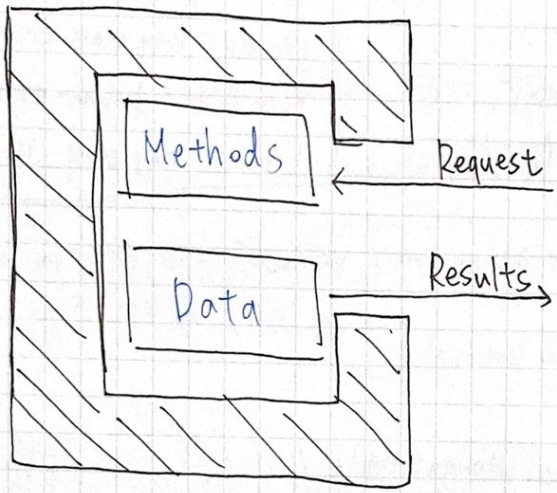
```
for (i=1 to aList.genLength()-1 ) {
    aList.retrieve (1, dataItem, success);
    aList.remove (1, success);                    先刪除後插入
    aList.insert (aList.getLength()-i+2, dataItem, success);
}//for
```

(二) reverseList2 (in aList: List, out source: boolean)

```
for(i=1 to aList.getLength()-1 ) {
    aList.retrieve(aList.getLength(), dataItem, success);
    aList.insert( i, dataItem, success);          先插入後刪除
    aList.remove (aList.getLength()+1, success);
}//for
```

- C++類別 Cpp Class.

□ Encapsulation combines an ADT's data with its operation to from an object. 封裝.



□ Inheritance:

- A derived class or subclass inherits any of the publicly defined methods or data members of a base class or superclass or subclass
  父類別 子類別

```
#include "Sphere.h"
enum Color { RED, BLUE, GREEN, YELLOW }
class ColoredSphere : public Sphere
{ public:
    ... Color getColor() const;
  private:
    Color c;
}
```

父類別: Sphere
子類別: ColorSphere.

- An instance of a derived class is considered to also be an instance of the base class.
- An instance of a derived class can invoke public methods of the base class.

- Cpp Overloading　　C++類別的多載

```
class Rational {
  protected:
    long numerator;
    long denominator;
    void reduce(void);
  public: ...
};
class Integer: public Rational {
  public:
    void setRational(long,long); // overriding 覆載
    void setRational(long);      // overloading 多載
};
```

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

- C++ NameSpaces. 命名空間
  - A mechanism for logically grouping declarations and definitions into a common declarative region.
  - The contents of the namespace can be accessed by code inside or outside the namespace.
- C++ Exceptions 例外處理.
  - A mechanism for handling an error during execution.
  - A function can indicate that an error has occurred by throwing an exception.
  - The code that deals with the exception is said to handle it.

單元3: 鏈結串列

- 基本架構

  → 有 head, 有門擋住 tail

△ programming with linked list 指標

△ pointer - based Implementations 鏈結串列

  ↳ contains the location, or address.

  指標 = 門牌 (int *p)

△ *p : memory cell.

△ p = &x ⇒ &x = 房子x的門牌.

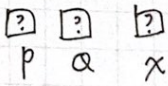
△ need to add "p = new int"

  (or 發現記憶体配置 error)

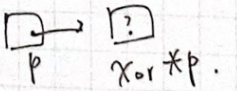
△ run 完要 delete (設為 NULL)

- EXAMPLE:

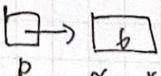
(a) Declaring pointer variable: int *p, *q; int x;



(b) pointing to statically allocated memory: p = &x;

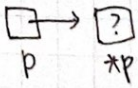


(c) assigning a value: *p = 6

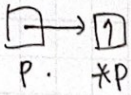


(d) allocating memory dynamically

p = new int



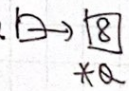
(e) assigning a value: *p = 7



(f) allocating memory dynamically & assigning a value

Q = new int

Q = 8



(g) copying a pointer: Q = P