DS 筆記

1-1. 遞迴的基本概念

① 把問題縮的越來越小

② 問題變小還是同一個問題，所以可以用同一個 function

=> 精簡化程式，讓別人好理解、debug

1-2. 反向印出字串

*string.substr(index, length)

Key point：問題簡化

Ex: 字串的字數漸漸減一 => 問題簡化

最後有空字串 => 終止條件 (base case)

* 遞迴出來的結果相同，但程式者需思考何者更適合使用者

1-3 GCD

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

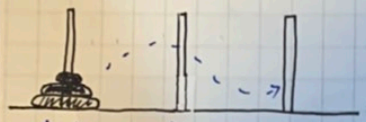
```
gcd2(int x, int y){
    if!(x%y) return y;
    else     return gcd2(y, x%y)
}
```

1-4 找第 K 大(小)

① 用類 binary search → 只處理其中一邊 more efficient

② Linear Recursion：線性遞迴(只擇一返回)

1-8 河内塔介绍



Key point : $f(n) \rightarrow f(n-1)$

```
solveTowers ( count , source , destination , spare )
  if ( count is 1 )
     Move a disk Irectly from source to destination => 把最後一層解決
  else {
     solveTowers ( count -1 , source , spare , destination ) ; => 把終点當輔助
     solveTowers ( 1 , source , destination , spare ) ;
     solveTowers ( count -1 , spare , destination , source ) ; => 把起点當輔助
  }
```

1-10 Binary Recursion

```
drawTicks ( length )
  if ( length > 0 )
     drawTics ( length -1 )

  draw tick of the length

     drawTics ( length -1 )
```

```
int sum ( int a, int b)
{ .... a+ sum ( a+1 , b);
} // 8 call
```

vs

```
int sum ( int a , int n )
{ ... sum ( a , n/2 ) +
      sum ( a+ n/2 , n - n/2 ) ;
} // 15 call
```

1-12 Multiplying Rabbit (Fibonacci)

① 可用第 0 項

Better :

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

1-14 指數

迴圈

```
double power1( double x , int n)
    double result = 1 ;
    while ( n > 0)
        result = x * x ;
        n - - ;
    return result;
```

```
double power2 ( double x , int n )
    if ( n == 1 )
        return x ;
    else
        return x * power1( x , n-1 );
```

1-17 n 選 k 個組合數

Base cases : c(k,k) = 1 or c(n,0)

If k > n => c(n,k) = 0

有幾種選法 = 有多少 Base cases

For example : c(4,2)

Base Cases = c(2,0) 、c(1,0) 、c(1,1) 、c(2,2) 、c(2,2) 、c(2,2)

Non-Base Cases = c(n,k) - 1

Tail Recursion = 可以轉為迴圈的 Recursion

=> 有時用 loop 相較於 Recursion 更加有效率

2-1 物件導向概念

Encapsulation (封裝):

hides inner details

有一群物件(相同、相似的)

用 Encapsulation → 更有效率、更不受外界影響

Inheritance (繼承):

reused

把之前寫過得東西(程式)

能用的直接 copy 過來

Polymorphism (多型):

在執行時,根據 Input 的不同

做出符合資料類別的結果

2-2 物件導向程式設計介紹

* Operation contracts:

Exceptions:

① Assume they never happen

② Ignore invalid situations

③ Return a value that singles a problem

④ Throw an exception

① Purpose ② Assumptions ③ Input ④ Output

1-3 資料抽象化原理　　ADT = Abstract Data Type

Modularity (模組化):

① 將程式切成一小塊一小塊

② Isolates errors

③ Eliminates redundancies (冗長)

→ Easier write、read、modify (修改)

內聚: 一個 function 別做太多事 (越少越好)

耦合: function 間傳遞的參數越少越好

Data abstraction:

不需要去管如何達成目的

只需要把要做什麼講清楚就好

補充:

```
reverse ( in aList : List , out source : boolean)

  for ( i = 1 to aList.getLength() - 1 ) {
     aList.retrieve ( 1, dataItem , success);   → 檢索
     aList.remove ( 1, success);
     aList.insert ( aList.getLength() - i + 2 , dataItem , success);
  } // for
```

1-8 C++ 實作

① Data members should be private.

→ Constructs can be public

② Public → 提供給外界使用, private → 外界看不到、用不到

③ 子類別可以繼承父類別的所有 methods (private 除外)

④ 外部看到的 position 要減一, 才是 implement 時的 index → 正確設計才對

→ 對外部來說, pos 不會為 0

2-16 C++ 命名空間，例外處理

- 可以把一堆 class 放在同一個 namespace 並使用 → 省時省力
- 容易出錯的地方：

try：去試看看 ⇒ throw ...

catch：把錯誤訊息保留下來

```
try { ... throw (type); ...
  catch (type1) {
    statement (s);
    ...
  } // catch()

  catch (type 2) {
    statement (s);
    ...
  } // catch()

  .
  .
  .
} // try()
```

⇒ 跟 switch 用法很像

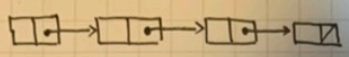
Chapter 3

- Link list :

首 □-□-□-□-□ 尾 X

Array ⟷ Link list

Preliminaries



指標 pointer
=> 不用移動資料 (array 不行)

- Pointer :

int *p declaration of integer pointer

(initially undefined , NOT NULL !)

p = &x (x 的門牌)

p = new int (申請一棟房子)

delete p (歸還房子)

p = NULL (門牌清空)

- 動態陣列

int arraysize = 50 ;

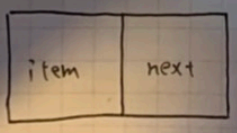
double *Anarray = new double[arraysize] ;

delete [] Anarray (歸還)

- Pointer - Base implementation (實用)
  - Public
    - isEmpty
    - getLength
    - insert
    - remove
    - retrieve
  - Private
    - find
  - Local variables
    - cur
    - pre

- A destructor is require for dynamically allocated memory

```
List :: ~ List
{
    while ( !isEmpty )
        remove ( );
}
```

- 


```
struct Node {
    int item ;
    Node *next ;
}
```

- Array v.s Pointer
  - Size : Linked list grow and shrink as necessary.
  - Storage : Array requires less memory.
  - retrieval : Array → 常數時間 (快), Linked List → 線性時間 (慢)
  - insert / delete : if 資料多, Array 較慢

- Restoring Linked list by using File :
  - ofstream outFile

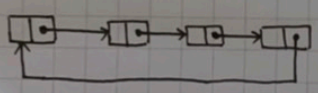
    Ex: outFile << cur->item << endl; → 寫檔

    outFile.close();
  - ifstream inFile

    Ex: inFile >> next item; → 讀檔

- Processing Linked list Recursively → 利用 head

- Variation (變化)
  - circular linked list



  - Double Linked list (双向指標)

```
{                (初始化：指向自己)
  Node * precde
  Node * next
}
```

- Application (Maintaining Inventory)
  - Local DVD store
  - operation on the inventory

- Dummy (Fake) head Node
  => Eliminate the special node !
  insert / delete

- 處理指標順序不宜更改

Chapter 4

- 定義語言 Defining Language

  x | y (x or y)

  xy or x·y (緊接著)

  - <number> = <digit><number> | <digit>
  - <digit> = 0 | 1 | 2 | 3 | 4 | ... | 9
  - <addition> = <digit> + <addition> | <digit>
  - 2位數值以上 <addition> = <number> + <addition> | <number>

- The Basic of grammar → recognition algorithm (辨識演算法)

  isId #遞迴　　終止條件：單一字元 / 遞迴呼叫：扣除最後一字元

  isId suffix

- Palindromes 回文

  ex. Anna , 38+83 = 121　　Base

  <pal> , empty string　<cn> | a<pal>a | b<pal>b | ... | z<pal>z

  <cn> = a | b | c | ... | z

- $A^n B^n$

  <Legal - word> = empty string | A <Legal - word> B

- Practice

  <S> = <L> | <D><S><S>
  <L> = <A><B>
  <D> = 1 | 2

  6[1. write all 3個字元字串? 1AA 1BB 1BA 2AA 2BB 2BA
  2. 包含多於3個字元字串? 12AAB (∞組)

- 猜謎至少一個字母所組成的字串，第一個字母大寫，others 小寫

- Algebraic Expression (代數)
  - Infix 中序運算式 ex: A + B
  - Prefix 前序 〃 ex: + A B
  - Postfix 後序 〃 ex: A B +
  - Advantage of prefix, postfix
    - No precedence rules 優先權
    - No association 結合律
    - No parenthese 括弧

- 中序轉前序 ((a+b)*c)

  => * + a b c

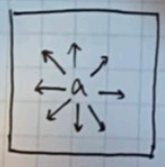
- Prefix
  - Grammar

    <prefix> = <identifier> | <operator><prefix><prefix>
  - recursive recognition
    - Base Case: One lowercase is a prefix exp
    - Recursive: <operator><prefix><prefix>

* 重要特性: 一個前序式後面再接上非空字串不一定是前序式

- Back tracking (回溯)

  . 八皇后問題



- 64格
- 8前進方向

→ 避開死棋的位置

- Recursive
  - Base Case: 八欄填滿
  - Recursive Step: 續填其它欄位