My Notes

Important Concepts worth keeping

Today: / /

①

* interation 迴圈, recursion 遞迴 (不一定比較快)

* fractal 碎形

* 反向印出字串

→ 字串長度減一, Base case = 空字串 → do nothing

```
if (size > 0) {          |  if (size > 0) {
    cout ... ; (base)    |      recursion;
    recursion;           |      cout ... ; (base)
} // if                  |  } // if
```

→ C. B. A A. B. C

// size == 0 (base case) → do nothing

* s.substr (第幾個, 幾個字)

* GCD 最大公因數

→ if !(x%y) return y;

else return gcd2(y, x%y); // recursion.

* pivot 樞紐

* 分兩邊 → 計只要找一邊 → better

Punctuality: Showing esteem for others by doing the right things at the right time.

My Questions

Problems & Difficulties needing exploration

* linear recursion 線性遞迴 (較有效率)

→ 1. 檢查終止條件 2. 擇一遞迴 ex 找第k小

* Towers of Hanoi 河內塔

→ n个 ⇒ $2^n - 1$ 次 (3, 7, 15 ...)

→ Algorithm towers (num, start, end, temp) {

```
if (num == 1) cout << "move from" << start << "to" << end << endl;
else {
    towers(num-1, start, temp, end);
    cout << "move from" << start << "to" << end << endl;
    towers(num-1, temp, end, stard)
} // else
} towers
```

My Opinions

Thoughts, Inspirations, and suggestions

* recursion 呼叫次數 > 搬動次數.

* Binary recursion 二元遞迴

→ ex: int sumB (int a, int n) {

```
if (n==1) return a;
return sumB(a, n/2) + sumB(a+n/2, n-n/2);
} // sumB()
```

* 求兔子數量: rabbit(1)=1, rabbit(2)=1, rabbit(0)=0

→ rabbit(n) = rabbit(n-1) + rabbit(n-2); // like 費氏

守時：在對的時間，做對的事，來表明對別人的尊重。 《培基》

* 呼叫次数本身也是 recursion
* 费氏数列 $F_k = F_{k-1} + F_{k-2}$ ⇒ 呼叫次数指数成長
→ 電腦去當掉！

→ linearFibonacci(k) {
if(k==1) return(k,0)
else {(i,j) = linearFibonacci(k-1);
return(i+j, i); } // else
} // linearFibonacci() // 呼叫次数線性成長

* Acker(m,n) = n+1 if m=0
= Acker(m-1,1) if n=0
= Acker(m-1, Acker(m,n-1)) otherwise

* leaf nodes 葉節点, internal nodes 內部節点.
→ |leaf| - |internal| = 1 ⇒ 数量差1

* Tail recursion 尾端遞迴
→ 在尾端進行遞回, do nothing after recursion
→ 可轉為迴圈 → 比較有效率
* Sum: 1. 遞迴定義 2. 問題簡化 3. 終止條件 4. 保證終止

My Notes
Important Concepts worth keeping

Today: / /

②

- Attributes = data members
- Behaviors = methods
- Encapsulation(封裝)：hides inner details.
- Inheritance (繼承) = reused. (類似但不相同)
- Polymorphism(多型) - determine appropriate operations.

Operation Contracts 運算合約

1. Purpose 目的
2. Assumptions 假設
3. Input 輸入
4. Output 輸出

Abstract Data Type (ADT) = motives

* Modularity - 切割程式 → 有系統管理. error少. 重複性低
1. Cohesion 內聚 → 高 = 每個function只做一件事
2. Coupling 耦合 → 低 = function間傳遞參數少

* Functional abstraction 功能性的抽象化 → specification + implementation

Information hiding = Hides certain implementation

* Data abstraction 資料抽象化

* retrive 取得

My Questions
Problems & Difficulties needing exploration

* (query) → 不會改變只會search.
* predecessor 先行者 successor 後繼者
* create . destroy . IsEmpty . getLength . insert . remove . retrieve

ex: reverseList (in aList=List, out source=boolean) {

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

} // reverse()

* 新增 刪除 檢索

* An object is an instance of a class

My Opinions
Thoughts, inspirations, and suggestions

⇒ class defines new data type.

* class contains data members and methods → { public / private

* defination = header file 描述 Classname.h
* implement = implementation file 實作 Classname.cpp
* class 的 default = private
* struct的default = public

My Notes

Important Concepts worth keeping

Today: / /

* constructors 建構子 - default 和 class 同名. no return. not void.

→ compiler will generator a default one if not defined.

→ a class can have several constructors.

* classname :: method : initializer

* Destructors 解構

* class ColoredSphere : public Sphere (繼承 - 可多个)

→ 父類別 - Sphere ; 子類別 - ColoredSphere

(base class) (derived class)

* protected - subclass instances.

→ 繼承者可使用

* overriding 覆載 (繼承間相同. 連參数都一樣) 以時叫為主

* overloading 多載 (名稱相同. 參数列不同)

* 資作 - 從0開始. 描述 - 從1開始

* namespace 命名空間. scope 範圍

* Exception 例外處理

→ 設定保護範圍 - try { ... throw (type); ... }

→ 捕捉例外狀況 - catch (type1) { statement(s); }

Let your Yes be Yes, and your No be No. 2:...

《Mathew》

My Notes

Important Concepts worth keeping

Today: / /

(3)

* 陣列 → 需要移動資料
  鏈結串列 → 不需移動資料
* 指標 = 門牌 (int *p)
* p = new int ; // The new operator
  p = &x ; // The address-of operator &
* Dynamic allocation 動態配置
* std::bad_alloc → the operator new cannot allocate memory
* delete p;
  p = NULL;
* memory leak → p有指向來面時使其=NULL
* 動態(配置)陣列:

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

→ 陣列名稱 = 指標

* 配置更大空間:

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

→ 需用for搬移

My Questions

Problems & Difficulties needing exploration

* delete [] oldArray; // 記得 delete
* #include <cstdio>

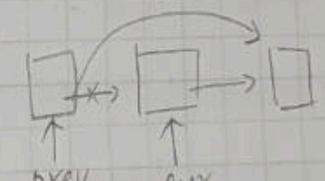
```
int main() {
  ⋮
  FILE *outfile = NULL;
  outfile = fopen (filename.c_str(),"a");
  if (outfile != NULL) ...
  ⋮
} // main
```

* 結構 Struct
* If(head == NULL) → 要建的
* head = new node; head = NULL → memory leak

My Opinions

Thoughts, inspirations, and suggestions

* 刪除 prev->next = prev->next->next



if 刪第1个 ⇒ head 要特殊處理

* while (!isEmpty()) remove() // 釋放 memory
* deep copy 深層複製 * shallow copy 淺層複製

	size	space	time
Linkedlist	動態	費	慢
array	靜態	省	快

My Notes

Important Concepts worth keeping

Today: / /

* Linkedlist save = 只存資料.不留指標
* 加入 tail 指向尾端
* ofstream ⇒ outputfile... 宣告輸出檔案

  file << data ⇒ 逐一寫檔.只存資料 * 記得 close();
* ifstream ⇒ inputfile... 讀檔
* 開完內的資料也可以是一個物件

My Notes

Important Concepts worth keeping

Today : / /

My Qu

Problems & D

④

* X | Y = X or Y ⇒ 或

  XY or X · Y ⇒ 緊接著

* recognition 辨識演算法

  → if it is recursive → easier

  * 以遞迴定義語言之回文

  * 38+83=121 ; 165+561=726 , 726+627=1353 1353+3531=4884 #

  * Infix 中序 ; Prefix 前序 ; Postfix 後序

  * Infix = 完整括弧 = <infix> = (<infix> <operator> <infix>)

  * Prefix = 左括弧 = 中轉前 ⇒ ((a+b)*c) → *+abc

  * Postfix = 右括弧 = 中轉後 ⇒ ((a+b)*c) → ab+c*

  * 一個前序式加上一個非空字串 ⇒ 一定不是前序式 #

  * 最長前序式的結尾必須是該字串的結尾

  * 八皇后問題 → 遇到僵局就退回 (backtrack)

  * 最簡 → 假設 → 歸納 (遞迴與數學歸納法)

* 學習心得：

1~4章元中許多練習題都是我們曾經寫過的，老師在上課中都可以提出各種不同的解法、真的很厲害，也讓我們能找到更好、更完美的寫法。

If you don't know where you're going it doesn't matter what path you take.

– Lewis Carroll