#11027245

遞迴:

思考核心:使問題越來越小(解析問題，使問題單一化並能重複解構)

優點:寫程式快速,容易解釋,看懂

缺點:容易造成系統負擔，效率低

遞迴定義

遞迴呼叫必須減小問題

完整的結束條件

確定每個問題都能終止(避免無窮)

遞迴例子：

一元遞迴:

1.找第 K 小的數值(選出一點為基準,大左小右,接下來只需要找其中一邊,結束條件:一點為第 K 小的數值)

2.河內塔(將小盤子搬到輔助的竿子上,將最大盤搬到另一根柱子,將小盤子搬到大盤上,結束條件:盤子為一個)

3.指數函數計算(用二分法計算較有效率,分成指數為奇或偶)

4.排列,組合方法

二元遞迴:(不一定會比一元遞迴快)

1.畫尺刻度(畫小一個長度的遞迴,畫自己長度的尺,畫小一個長度的遞迴,結束條件:長度為 1)

2.費氏數列(會有算過的函式被重複呼叫,可以用空間換時間,將算過的函式存起來,就不須再算一次)

尾端遞迴:函式的最後一個指令是呼叫遞迴

可以直接改寫為迴圈的形式

寫 CAL 時有練習過遞迴,有熟悉它的用法，後來在寫 project 的時候我有想刻意將能不用到遞迴的程式就用迴圈寫，其實也不太知道那些地方是實際遞迴比較好運用的，透過這次上課知道了河內塔，費氏數列等透過遞迴寫法會更好理解的 code，畢竟好的 code 並不是效率最高的 code 而是其他工程師看得懂，好維護的 code

寫程式應該:

1.模組化

2.程式風格

3.有修改空間

4.容易使用

5.發生錯誤也保證安全

6.能 debug

7.測試

物件導向(OOP)

運算合約:

1.目的 2.假設 3.輸入 4.輸出

資料抽象化(ADT): 優點:容易讀寫與修改

必須有描述與實作(隱藏資訊),限制程式對其直接存取(只能用給的 operations)

設計 ADT:1.屬性(想解決的問題屬性?) 2.運算(用怎麼樣的運算比較好?/需要甚麼樣的運算?)

建構子(constructor):(可以不只一個,因應不同需求)

1.名子跟 class 的名子一樣

2.沒有 return type

解構子(destructor):(歸還記憶體,防止 memory leak)

繼承(inherit):在 class 名稱後寫: public 你要繼承的 class 名稱

能繼承歸在 public 裡面的東西

ex: class ColoredSphere: public Sphere

ColoredSphere 為 Sphere 的子類別(derived class)

多載(overloading):

method 名稱一樣但傳入參數資料型態(參數列)不同

覆載(overriding):(繼承時才能使用)

method 名稱和參數列跟父類別一樣(呼叫時依物件選擇 method)

using namespace 使用命名空間 使用範圍解析::去觸及裡面的物件

命名空間 std : C++ Standard Library

exceptions(例外處理)也是一個 class

try{ throw(type); } 發生問題時丟出例外(需限定例外規格)

** error debug 時好用的東西*

catch(type x){} 接收例外嘗試修補(可以集中寫)

catch(type y){}

predecessor(前面一個)

successor(後面一個)

優解:

高內聚:盡可能地讓每個函式只做一件事情

低耦合:盡可能的減少傳入函式的參數

這章講了很多物件導向的概念,其中繼承,覆載和解構子對我來說算是新知識，對裡面比較有興趣的就是 throw 這一塊，這個語法之前沒有深究過，但感覺就是一個分析跟 debug 的神器!

鏈結串列(Linked list):

優點:動態存取(不需要搬動資料,容易增減資料),資料可以不用在連續位置

缺點:必要花的空間較多(多了存指標的空間),拿取資料比較慢(需要走訪)

每個節點會有資料和 next(指到下一個節點的位置,但最後一個要指到 NULL)

基本功能:新增刪除(在頭會是 special case),走訪

淺層複製(shallow copy):僅 copy 頭的位址(可存取同個 linked list 的資料)

深層複製(deep copy):複製整個鏈結串列(新的)

存檔:只存資料(走訪並逐一寫入,不必寫入指標)

重建:使用 tail,每抓到一個資料就 new 一個新的節點並把資料寫入節點

變形:(可以實作在一起)

環狀鏈結串列(circular):在尾端指向 head(能重複走訪)

dummy head node(在 head 前面新增一個無用節點,可以去除一些 special case ex:新增,刪除)

雙向鏈結串列(doubly linked list):有指向前一個節點的指標(功能性好,但要花更多空間與時間維護)

*須更注意指標連結順序

用鏈接串列串鏈結串列

陣列版鏈結串列

有 head 和 free(存放陣列中可以使用的空間,類似 new 配給的記憶體)

也有 item 和 next(兩個值)

多項式的鏈結串列:(加法)

存係數和指數(還有 next)

可以依兩個鏈結串列指數的大小(a>b, a==b, a<b)不同做不同的事

最後一邊為空時並把剩下的接完

指標:

(型別*) 變數名稱 宣告指標

&變數名稱 拿取位址

*指標 指標指的內容

指標 = new 型別 配置記憶體(如果沒有 memory 會丟出 exception)

不用時要 delete(防止 memory leak) (並通常會讓指標為 NULL,防止動到不該動的地方)

動態陣列:

型別* 變數名稱 = new 變數名稱[arraysize] (指標會指向此陣列的頭)

指標名稱[k] 等同於 *(指標名稱+k)

變數名稱 = new 型別[x * arraysize] (要求更大空間)

對指標停留在有概念不會活用的階段，只有用過單向的鏈結串列,並且不太熟悉雙向的鏈結串列對我來說簡直是打開一個新世界，用 dommy head node 的概念來解決鏈結串列的 special case 也是一個不錯的啟發。

定義語言

x|y means x or y

xy means x followed by y

C++ identifier grammer(不只有一個表示法)

<identifier> = <letter>|<identifier><letter>|<identifier><digit>

<letter> = a|b|...|z||A|B|...|Z|

<digit> = 0|1|...|9

(單一的 letter 為遞迴的 base case)

例子:

回文定義

<pal> = empty string|<ch>|<ch><pal><ch>

<ch> = a|b|...|z|A|B|...|Z

<S> = <L> | <D><S><S>

<L> = A|B

<D> = 1|2

包含三個字元的字串 : 1AA, 1AB, 1BA, 1BB, 2AA, 2AB, 2BA, 2BB

包含三個字元以上的字串: 11AAAA

至少一個字母組成的字串語言,第一個字母是大寫其餘則為小寫(用遞迴文法描述)

<S> = <U>|<S><L>

<U> = A|B|...|Z

<L> = a|b|...|z

運算式

前,中,後序運算式(依據運算子在運算元的哪裡)

後序式的優點:不用加上括號或定義沒有括號時的優先順序,只要從左到右依序計算

中序式轉(前)後序式:

完整加入括號並將運算子移到括號(左)右邊

後序式定義:

<postfix> = <identifier> | <postfix><postfix><operator>

<operator> = +|-|*|/

<identifier> = a|b|...|z

回朔(backtracking):

八皇后問題:

終止條件:填滿八個欄位

遞迴呼叫:填下一個欄位,如果填不了則回朔,嘗試下一種可能

航班問題:抵達目的地(起點到終點有無航班)

base case:起點(目前的點)等於終點

去每個相鄰且沒有走過的城市(確認有無航班以及重複的城市)

數學歸納法:

假設 base case

假設在 x=n 對,證明在 x=n+1 也對(歸納)

有理解到中序式跟後序式的差異，也稍微理解電腦是怎麼處理(理解)這一塊計算

方面的問題，遞迴的部分也理解到分析解構問題的重要性。