電資二 10920137 柯文凱

遞迴 Recursion

何謂遞迴? ->用鏡子照鏡子會使鏡面中的鏡子越來越小 -->利用重複的問答把問題給縮小 好處: 程式精簡化，人可以快速判讀程式再做什麼 壞處: 可能會使程式運行速度變慢 運用:階層、搜尋、河內塔、費式數列

二階遞迴 Binary Recursion

利用遞迴包遞迴的方式，可以做出更細節的需求 像是幫一定長度的尺畫上刻度

河內塔 Hanoi Tower

由 André Karwath aka Aka - 自己的作品, CC BY-SA 2.5, 維基百科

規則

有三根杆子A、B、C，A杆上有N個(N>1)穿孔圓盤，盤的尺寸由下到上依次變小。要求按下列規則將所有圓盤移至C杆： 1.每次只能移動一個圓盤 2.大盤不能疊在小盤上面 解法 : 把A塔頂部的N-1塊盤移動到B塔，再把A塔剩下的大盤移到C，最後把B塔的N-1塊盤移到 C

程式碼

```
void Towers(int n,char a,char b,char c){
    if(n==1){
        cout<<"Move disk "<<n<<" from"<<a<<" to "<<c<<endl;
    }
    else{
        Towers(n-1,a,c,b);
        cout<<"Move disk "<<n<<" from"<<a<<" to "<<c<<endl;
        Towers(n-1,b,a,c);
    }
}
```

費氏數列 Fibonacci Sequence

程式碼:

```
#include<iostream>
using namespace std;

 int f(int n)
{
    if( n==1 or n==2 )
        return 1;
    if( n >= 3 )
        return f(n-1)+f(n-2);
}

int main()
{
    int n;
    int d = 1 ;
    cin >> n ;
    while( n > d )
    {
        cout << f(d) << endl;
        d++ ;
    }

    return 0;
}
```

輸入: `8` 輸出: `1,1,2,3,5,8,13`

小結

遞迴在有一個固定公式、想法的時候可以快速寫出，但在重複多次之後速度可能比不上迴圈。

抽象化 Data Abstraction

什麼是抽象化? 舉個例子像是Google Map會判斷交通告訴你路該怎麼走，大概花費多少時間，但你不用知道地形長怎樣，你的交通工具是如何運作的，路上到底有幾輛車讓地圖顯示這裡塞車，只需要知道怎麼用和會給你甚麼結果就好了。 而程式的Methods只會跟你說你會有啥結果，而不是告訴你如何得到。

Oriented Objective Programming 物件導向程式設計 是由許多**Class**物件來組成一個程式的 而Class物件的組成又由這些資料構成 * Data Members 用於儲存資料屬性(Attribute) * Methods / Member Funtions 程式運算的功能(Operations)

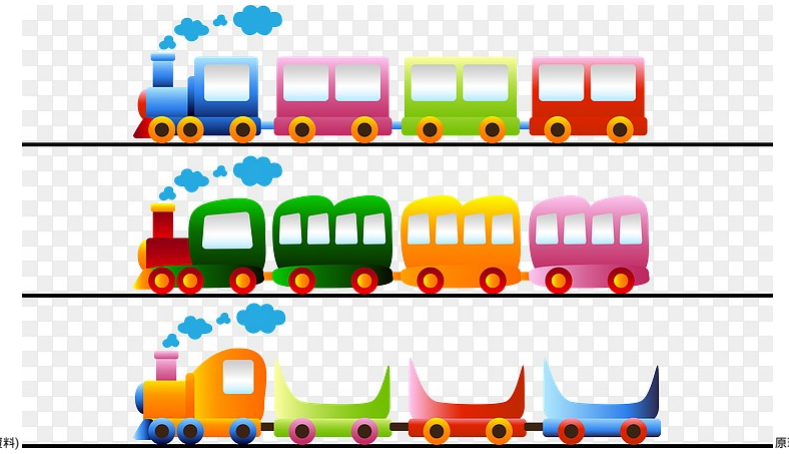
而Class所擁有的特性: * Encapsulation 封裝 ->把資料隱藏 不給外部直接接觸 * Inheritance 繼承 -> 在一定的權限下使用另外Class的Methods、Data 1. Private -> 只有該Class能夠影響 2. Protected -> (繼承)SubClass可以使用 3. Public - > 任何Class可以使用 * Polymorphism 多型 -> 讓一個程式提供 不同的功能

好處? 傳統的語言設計是把程式都當作一系列**函式**組成的，也就是說只有功能而已並不能個別儲存資料，但是物件導向語言的好處是**每一個物件都應該能夠接受資料、處理資料並將資料傳達給其它物件(Modularity 模組化)** 如此一來，程式變成一部份、一部份的程序，維護也變得簡單許多，設計也可以分區進行多工。

小結

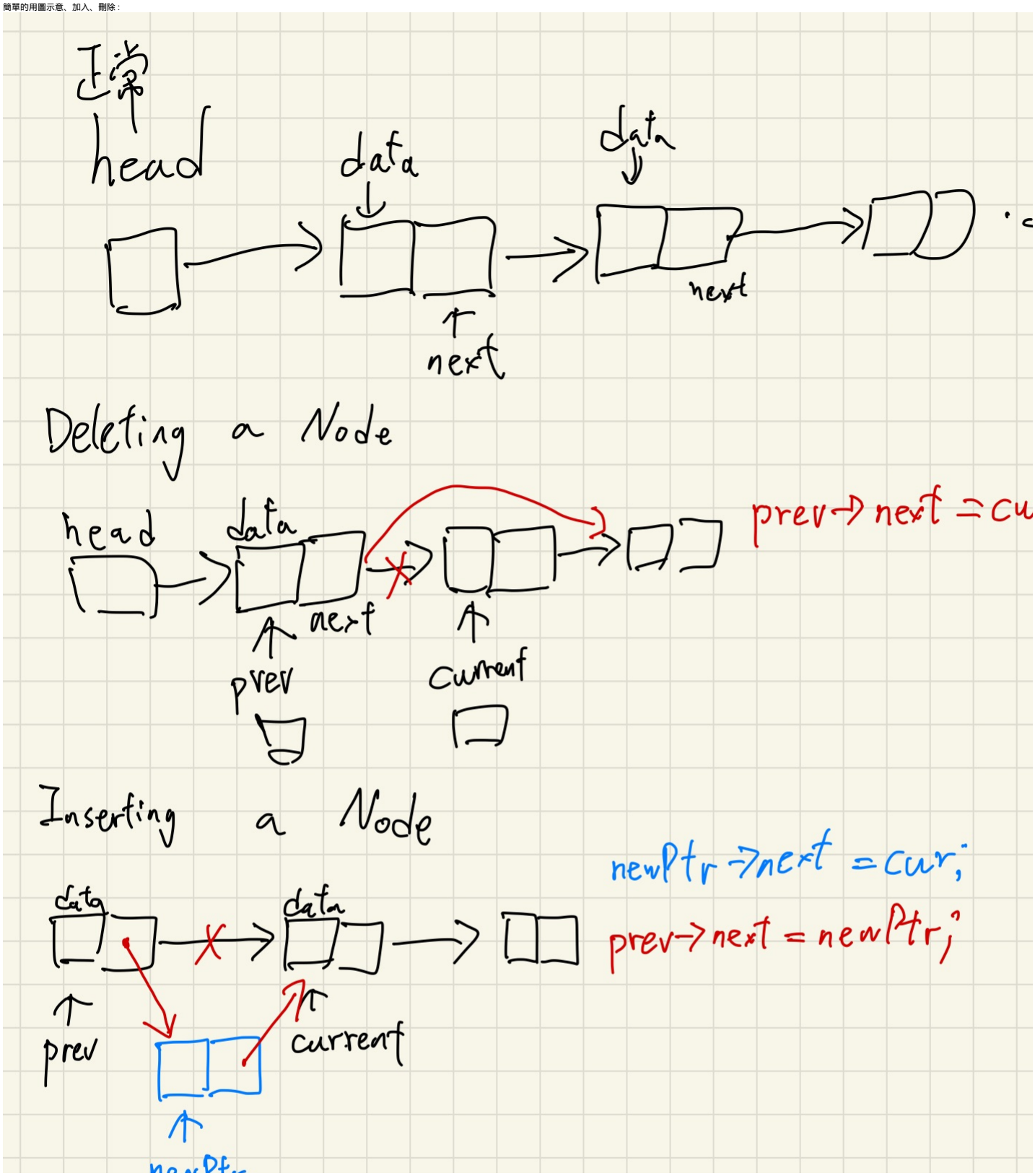
抽象化讓現今的程式語言可以持續發展，也可以比較直觀的讓人維護程式，除此之外幫一個Class創建初始化(Initialize)和解構(Deconstruct)是一個好習慣，以免資料過多!還有方便卻要小心使用的 `@Overide` `@Overload`

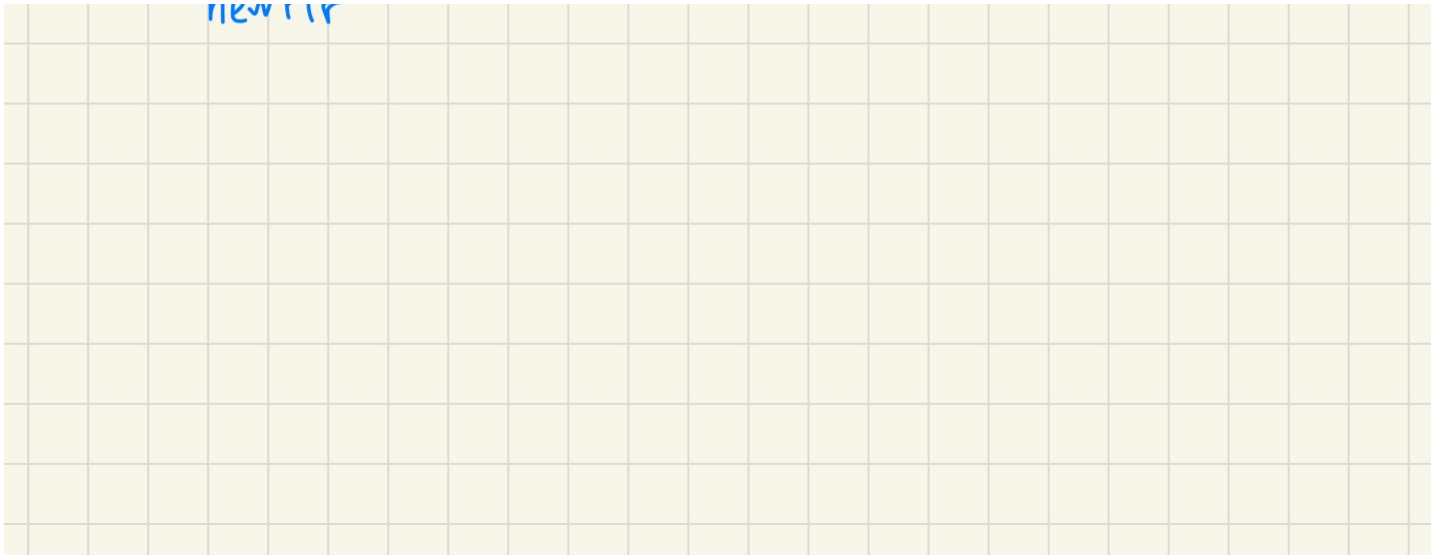
串列 Linked list



串列就像火車，一節接著一節上面載著不同的東西，一旦有一台斷軌了之後，就找不到後面的車了(資料) 原理:使用**指標(Pointer)** 指著下一個資料的地址再由下個資料指下一個資料的地址....

簡單的用圖示意、加入、刪除 :





使用串列時Head及Tail需要注意 * 若Head的指標跑掉了-->後面資料遺失-->Memory Leak * 若Tail沒有在最後的位置-->新的資料佔據舊資料的地址-->Memory Leak

使用指標時須注意的好習慣!!!!! 否則將會在無形之中吃著你的記憶體(Memory Leak)...

```
delete p ; // 清空房子
p = NULL ; // 忘記地址!!
```

Array 動態陣列

```
int size = 50 ;
double *array = new double[size] ; // 可以儲存50個double!
```

如果要移動資料只能一個一個移動，較麻煩及費時。 刪除舊地址需加[] ex: `delete [] oldArray;`

List 串列 在單元二介紹過

std::list 是一個c++裡面有的函式庫使用時需要在headfile宣告 `include <list>` 是一個**Double Linked List**和指標不一樣的式list的儲存方式是不支援隨機存取的功能的，運作龐大的資料可能需要較長時間但有提供許多預設的功能可以使用

```
push_back: 把一個元素添加到尾端
push_front: 把一個元素插入到頭端
pop_back: 移除最後一個元素(尾端)
pop_front: 移除第一個元素(頭端)
insert: 插入元素
erase: 移除某個位置元素，也可以移除某一段範圍的元素
clear: 清空容器裡所有元素
```

小結

在剛接觸程式語言時，指標的觀念可能是最模糊的，不太確定他如何運作，到底是指著地址還是存著，常常分不清楚，但在後續的接觸之後現在有需多強大的函式庫提供像式Vector、list、Stack之類...讓我們省下許多不必要的麻煩~