Choosing k out of n things

The number of ways to choose k-1 out of n-1 things and the number of ways to choose k out of n-1 things

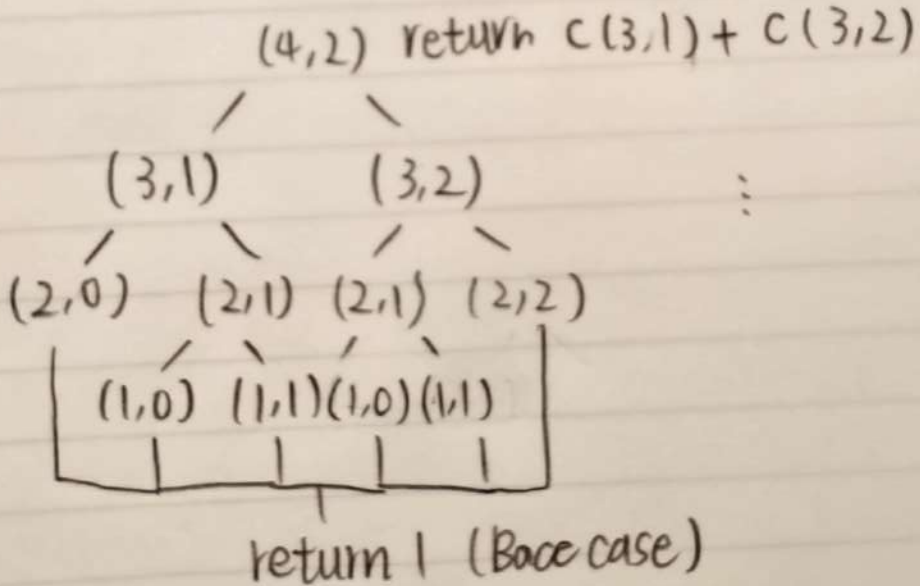
$\Rightarrow C(n,k) = C(n-1,k-1) + C(n-1,k)$

Base case

$C(k,k)=1$ $\quad \Rightarrow 0<k<n$

$C(n,0)=1$

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



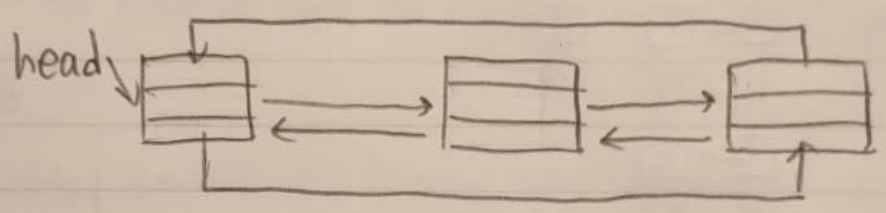
Doubly linked lists

1. Each node points to its predecessor (前者) and its successor (後者)
2. always has a 'dummy head (掛名的)
3. often circular to eliminate special (use)

code:

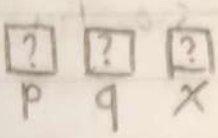
```
Struct item{
    int item;
    Node *pre;
    Node *next;
};
```

示意图:

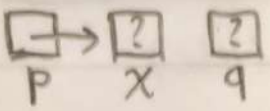


Pointers

```
int *p, *q;
int x;
```

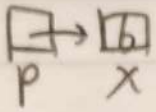



```
p = &x;
```

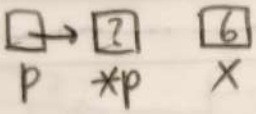


(x's memory location)

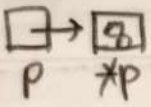
```
*p = 6;
```



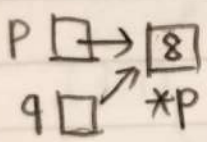

```
p = new int;
```



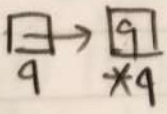

```
*p = 8;
```



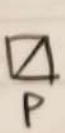

```
q = p;
```




```
q = new int;
*q = 9;
```



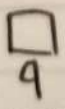

```
p = NULL
```



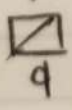
8 ⇒ memory leak

To prevent memory leak

⇒ `delete q;`



`q = NULL;`



Recursion 遞迴

Towers of Hanoi 河內塔　　移 2^n-1 次，n=層數

code:

```
Void SolveTowers(int count, char sou, char des, char spave){
    if(count==1)
      cout<<"move top disk from pole"<<sou
          <<"to pole"<<des<<endl; // base case
    else{
      SolveTowers(count-1, sou, spave, des);
      SolveTowers(  1  , sou, des, spave);
      SolveTowers(count-1, des, sou, spave);
    }
```

sou　　spave

// 移起點的 count-1 個到中繼站

// 移 sou 的 1 個到終點 → des

// 移 spave 的 count-1 個到 des

classes of objects (called instances)
Attributes: data members
Behaviors: methods

Encapsulation 封裝: hide inner details

Inheritance 繼承: reused

polymorphism 多型

An ADT is composed of
- A collection of data
- A set of operations on that data

Specifications Implementation

overloading 重載 ex: void function (int a)
void function (char b)

constractors
1. 新增並初始化 instances
2. 與 class 同名
3. no return vaule
4. compiler 會自動作一個
5. can include many constractors

Destractors
1. destory instane when lifetime ends
2. same as constractors
4.