遞迴

階層，最大公因數，搜尋，費氏數列，河內塔

最大公因數

x=9, y=6 取最大公因數

(法一) x<y

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

(法二) x≥y

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

找第k小的數

```
kSmall (k, anArray, first, last)
  = kSmall (k, anArray, first, pivotIndex -1)
                    if (k < pivotIndex - first +1)
```

左

樞紐

```
  = p   if (k = pivotIndex - first +1)
  = kSmall (k -(pivotIndex - first +1), anArray,
            pivotIndex +1, last)
                    if ( k> pivotIndex-first+1)
```

右

顛倒印

```
if (low > high) then
   Swap anArray[low] and anArray[high]
   ReverseArray (anArray, low+1, high-1)
   return
```

河內塔

```
solveTowers (count, source, destination, space)
   if (count is 1)
      Make a disk directly from source to destination.
   else {
      solveTowers (count-1, source, space, destination)
      solveTowers (1, source, destination, space)
      solveTowers (count-1, space, destination, source)
   } // else
```

求 rabit number

Base cases : rabbit(2), rabbit(1)

Recursive : rabbit(n) = rabbit(n-1) + rabbit(n-2) ; if(n>2)

費氏數列: 1 1 2 3 5 8 13 21 34 55 89

```
int rabbit (int n) {
   if (n <= 2) return 1;
   else return rabbit(n-1) + rabbit(n-2);
} // rabbit()
```

費氏數列

Algorithm linearFibonacci(k)

Input: A nonnegative integer k

Output: Pair of Fibonacci numbers (F_k, F_{k-1})

if (k=1) return (k, 0) // (F_1, F_0)

else

(i, j) = linearFibonacci(k-1) // (F_{k-1}, F_{k-2})

return (i+j, i) // $(F_k = F_{k-1} + F_{k-2}, F_{k-1})$

遊行隊伍排列數

花車 F(n) 樂隊 B(n) 不能同時有兩樂隊

$P(n) = F(n) + B(n)$

花車殿後 $F(n) = P(n-1)$

樂隊 : $B(n) = F(n-1) = P(n-2)$

Solution: $P(1) = 2$

$P(2) = 3$

$P(n) = P(n-1) + P(n-2)$ for $(n > 2)$

二元樹

|leaf nodes| - |internal node| = 1

葉節点 內部節点

尾端遞迴 = 可change成迴圈

catch block (捕捉例外狀況)

```
catch (ExceptionClass, identifier) {
    statement(s);
} // catch
```

Throwing exceptions (例外狀況的類別)

```
void myMethod(int x) throw (MyException){
    if (...)
        throw MyException("MyException...");
    ...
}//
```

Data Abstraction 抽像化

物件導向：classes of objects 封裝、繼承、多型

Encapsulation: hides inner details

Inheritance: reused copy

Polymorphism:

Operation Contracts 運算合約

purpose Assumptions Input Output

key Issue: Modularity, Style, Modifiability, Ease of Use
Fail-safe programming, Debugging, Testing

Modularity 模組化

Abstract Data Types: motives

highly cohesive 高內聚 loosely coupled 低耦合

Functional abstraction 功能性抽象化

Information hiding

Typical operations on data (Add, Remove, Ask)

Predecessor 先行者

successor 後繼者

insert 插入 remove 刪除 retrieve 檢索

aList.createList() aList.insert(1, milk, success)

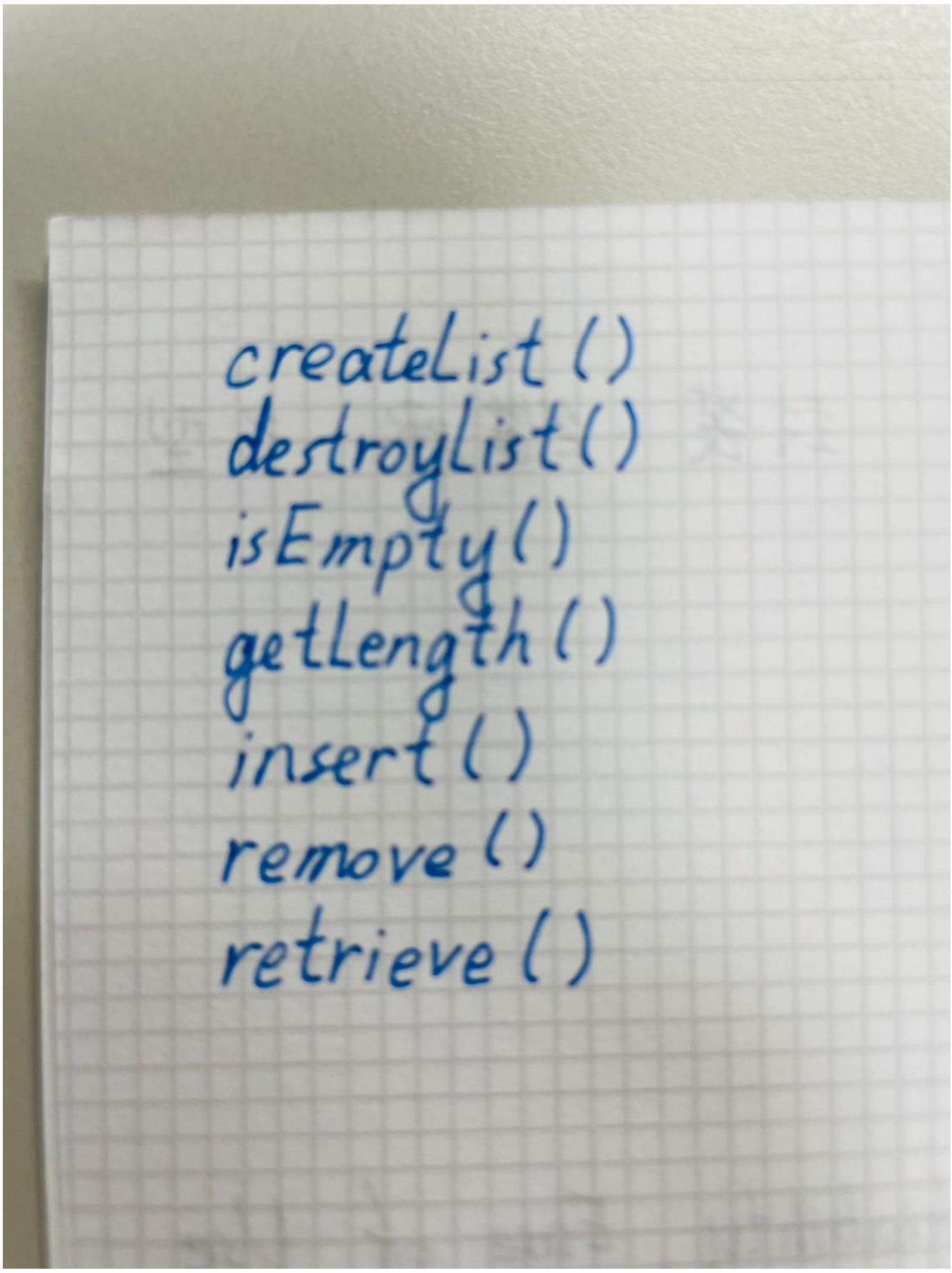
aList.destroyList() 原位往後

isEmpty(): boolean {query}

getLength(): integer {query}

aList.remove(3, success)

刪除位3並往前移


createList()
destroyList()
isEmpty()
getLength()
insert()
remove()
retrieve()

```
displayList (in aList:List)
for (position = 1 to aList.getLength())
{ aList.retrieve (position, dataItem, success);
  Display dataItem;
}// end for
```

置換

```
replace (in aList:List, in i:integer, in newItem:ListItemType,
         out source:boolean)
  aList.remove (i, success);          先刪後插入
  if (success)
    aList.insert (i, newItem, success);
```

反轉整个序列

```
reverseList (in aList:List, out source:boolean)
  for (i = 1 to aList.getLength() - 1)
  { aList.retrieve (1, dataItem, success);
    aList.remove (1, success);
    aList.insert (aList.getLength()-i+2, dataItem, success);
  }// end for
```

```
sortedIsEmpty(): boolean {query}
sortedGetLength(): integer {query}
```

新增 sortedInsert (in newItem: ListItemType, out success: boolean)

移除 sortedRemove (in index: integer, out success: boolean)

檢索 sortedRetrieve (in index: integer, out dataItem: ListItemType, ")

定位 locatePosition (in anItem: ListItemType, out isPresent: boolean): int

ADT operations

createAppointmentBook()　　建構

makeAppointment(···): boolean　新增

cancelAppointment(···): boolean　取消

isAppointment(···): boolean　是否有約

checkAppointment(···)　約會目的

改變約會日期或時間

```
read(oldDate, oldTime, newDate, newTime);
apptBook.checkAppointment(oldDate, oldTime, purpose);
if(purpose is not null){
  if(apptBook.isAppointment(newDate, newTime))
    write("There is an appointment at", newTime, "on", newDate);
  else{
    apptBook.cancelAppointment(oldDate, oldTime);
    if(apptBook.makeAppointment(newDate, newTime, purpose)
      write("The appointment has been moved to", newTime, "on", newDate);
  }//else
}// if
else
  write("There is no appointment at", oldTime, "on", oldDate);
```

改變約會目的

changeAppointmentPurpose

Implement ADTs

Private: only class

Protected: subclass

Public: any class

ADT Polynomial

Operation: degree()

coefficient (in power)

changCoefficient (in newCoefficient, in power)

Constructers (建構子)

1. Create and initialize new instances of a class
2. Have the same name as the class
3. Have no return type, not even void

Create a namespace (自訂命名空間)

```
namespace smallNamespace {
    int count = 0;
    void abc();
} // end smallNamespace
```

using a namespace (使用命名空間)

```
using namespace smallNamespace;
count += 1;
abc();
```

try block (設定保護範圍)

```
try {
    statement(5);
}
```

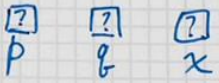
鏈結串列

Programming with Linked Lists (指標)
Pointer-based Implementations. (鏈結串列)
Circular Linked Lists (環狀鏈結串列)
Doubly Linked Lists (XX向鏈結串列)

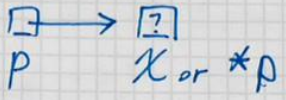
陣列(需要移動資料)
鏈結串列(不需要移動資料)

```
(int *)P;
 * Initially undefined, but not NULL
 * Static allocation (靜態)
p = &x
p = new int; (動態)
std :: bad_alloc( int the <new> header) 沒有空房
delete P;
P=NULL;
```

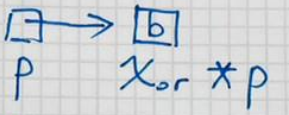
(a) int *p, *q;
int x;



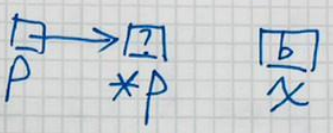
(b) p = &x



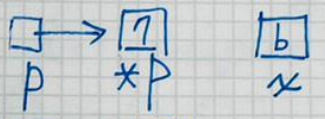
(c) *p = b;



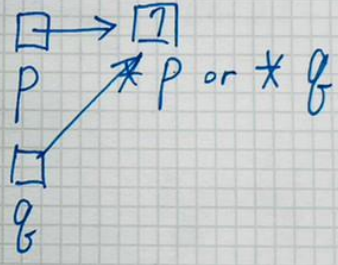
(d) p = new int;



(e) *p = 7;

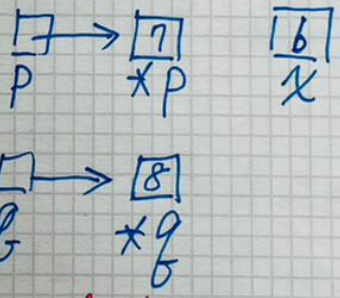


(f) q = p;

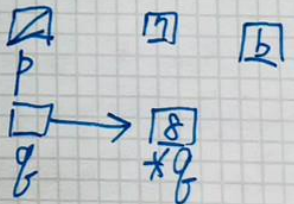


(g) q = new int;

*q = 8



(h) p = NULL; memory leak



(i) delete q;
q = NULL

動態(配置)陣列

```
int arraySize = 50;
double * anArray = new double [arraySize];
anArray[2] = *(anArray + 2)   陣列名 = 指標
```

配置更大的空間

```
double * oldArray = anArray;  // copy pointer
anArray = new double [3 * arraySize];
for (int index = 0, index < arraySize; index++) //copy old array
   anArray[index] oldArray[index];
delete [] oldArray;
oldArray = NULL;
```

```
# include <iostream>
# include <string>
# include <cstdio>

# define SID_LEN 12
# define SR_NUMS
using namespace std;
```

```
typedef struct student
{ char sid [SID_LEN];
  int scorce;
} // studentType
int main (void)
{FILE * outline = NULL;
 string fileName = "DSsample1.dat";
 studentType alls[SR_NUM]={
   {"  ", 60}  . . .
 outfile =  . . .
 if (outfile != NULL)
    saveFile (outfile, alls, SR_
 return 0;
}// end main
```

鏈結串列

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

It head is NULL, the linked list is empty

A node is dymatically allocated (節点)

```
Node * p ;
p = new Node;
```

刪除節点

```
prev -> next = cur -> next
prev -> next = prev -> next -> next
```

刪 first node

```
head = cur -> next;
cur -> next = NULL;
delete cur;
cur = NULL;
```

To insert a node between two nodes

```
newPtr -> next = cur
prev -> next = newPtr
```

To insert a node at the beginning of a linked list

```
newPtr -> next = head;
head = newPtr;
```

Inserting at the end of a linked list is not a special case

```
if cur is NULL
    newPtr -> next = cur;
    prev -> next = newPtr
```

走訪至欲增刪的節點

```
Node * prev, * cur;
for (prev = NULL, cur = head; (cur != NULL) &&
     (newValue > cur -> item); prev = cur, cur = cur -> next);

if (prev == NULL)
  head = cur -> next;
else
  prev -> next = cur -> next;
cur -> next = NULL;
delete cur;
cur = NULL;
if (prev == NULL) {
 newPtr -> next = head;
 head = newPtr;
} // if
else {
  newPtr -> next = cur;
  prev -> next = newPtr;
} // else
```

Public methods

- isEmpty
- getLength
- insert
- remove
- retrieve

Private method

- find

Private data members
-head 串列首
-size 節点數

陣列：size 固，少空間 常數時間，較快找到
指標：size 不固，多空間 線性時間，較慢找到
鏈結串列存檔：只存資料，不留指標
List 列表
Find 搜尋
Replace 置換
Insert 新增

* 以遞迴解題
grammer 文法，syntax 語法
recognition algorithm 辨識演算法
迴文：左→右和右→左都長一樣
Infix expression (中序運算式)
EX: a+b ((a+b)×c)/d
Prefix expression (前序運算式)
Ex: +ab /×+abcd
Postfix expression (後序運算式)
Ex. ab+ ab+c×d/

中序轉前序

Infix: ((a+b)×c)

Prefix: ×+abc

中序轉後序

Infix: ((a+b)×c)

Postfix: ab+c×

Advantages of prefix and postfix expression

- No precedence rules (優先權)
- No association rules (結合律)
- No parentheses (括号)
- Simple grammars
- Straightforward recognition (辨識) and evaluation (求解) algorithms

重要特性:

一个前序式後面再接上非空字串一定不是前序式

心得:

跟大一的計概比起來，資料結構的難度形成了一个很大的躍進，在看完教學影片後，雖然了解了code的架構，但仍有許多指令要自行到網路搜尋，並且反覆思考後才能融会貫通。經过這幾个星期的反覆練習，我漸漸的適應了在教学影片中学習概念，再上網尋找对應的程式指令，並且應用在任務或是挑戰上，在大一時，資訊幾乎都是來自CAL，然而升大二後反而發現網路上原來有那麼多可供使用的資料！