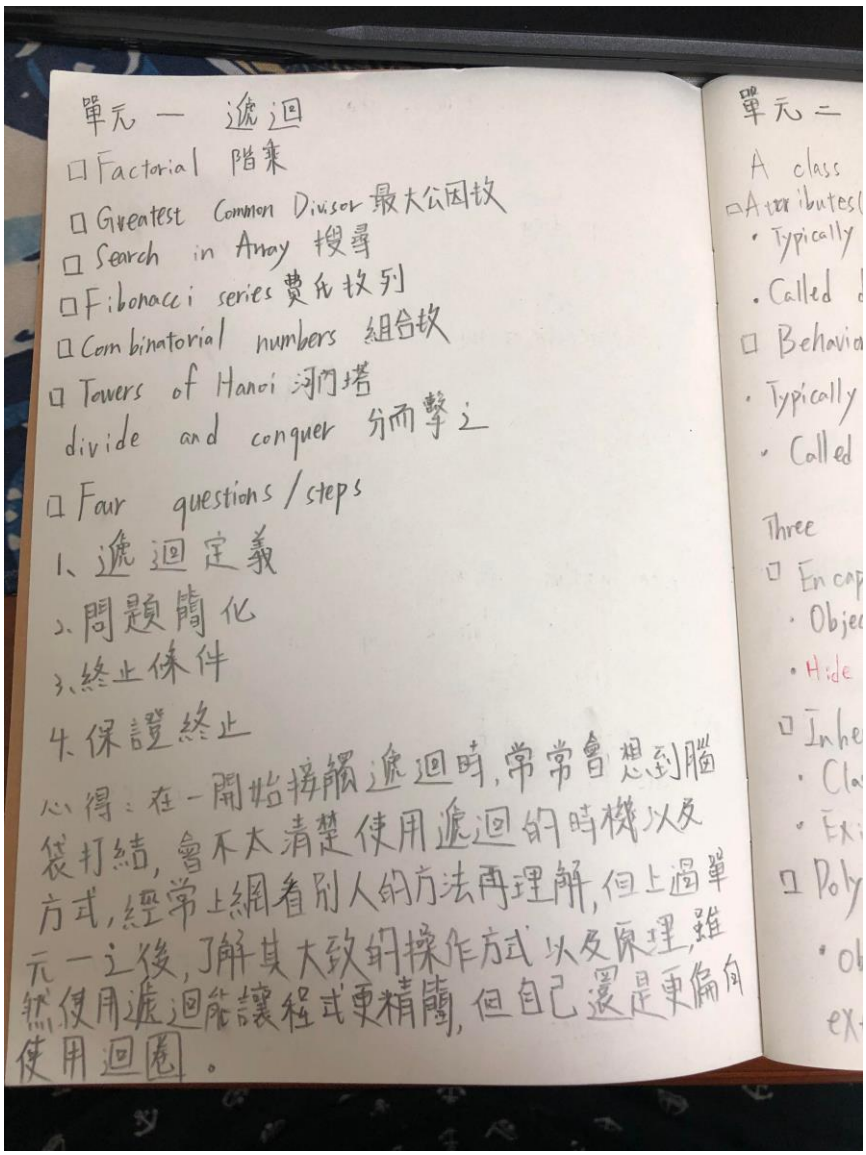


單元一 遞迴

□ Factorial 階乘
□ Greatest Common Divisor 最大公因故
□ Search in Array 搜尋
□ Fibonacci series 費氏故列
□ Combinatorial numbers 組合故
□ Towers of Hanoi 河內塔
divide and conquer 分而擊之

□ Four questions / steps
1、遞迴定義
2、問題簡化
3、終止條件
4、保證終止

心得：在一開始接觸遞迴時，常常會想到腦袋打結，會不太清楚使用遞迴的時機以及方式，經常上網看別人的方法再理解，但上過單元一之後，了解其大致的操作方式以及原理，雖然使用遞迴能讓程式更精簡，但自己還是更偏向使用迴圈。

單元二

A class
□ Attributes
• Typically
• Called
□ Behavio
• Typically
• Called

Three
□ En cap
• Objec
• Hide
□ Inhe
• Cla
• Ex
□ Poly
• ob
ex

單元二　資料抽象化

A class combines

- Attributes (characteristics) of objects of a single type
  - Typically data
  - Called data members
- Behaviors (operations)
  - Typically operate on the data
  - Called methods or member funtions

屬性

運算

Three characteristics

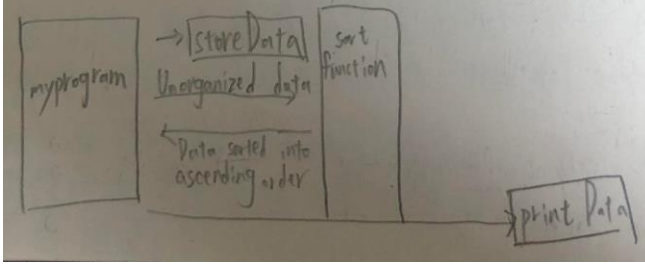
- Encapsulation 封裝
  - Objects combine data and operations
  - Hide inner detials
- Inheritance 繼承
  - Classes can inherit properties from other classes
  - Existing classes can be reused
- Polymorphis 多型
  - Object can determine appropriate operation at execution time.

□ key Issues in Programming

1. Modularity
2. Style
3. Modifiability
4. Ease of use
5. Fail-safe programming
6. Debugging
7. Testing

□ Cohesion - modules perform single well-defined tasks
- highly cohesive modules desired 高內聚

□ Coupling - measure of dependence among modules
- Loosely coupled modules desired 低耦合



Abstrac

myprogra

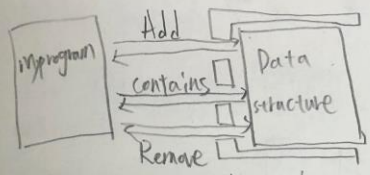
Compose

□ ADT

- Crea
- Dest
- Dete
- De
- Inse
- Dele
- Look

心
有很
程

Abstract Data Type (ADT)



Composed of < A collection of data
A set of operations on the data.

□ ADT List Operations
- Create an empty list 建構
- Destroy a list 解構
- Determine whether a list is empty 是否為空
- Determin the numbers of items in a list 計算個數
- Insert item in a given position in list 插入
- Delete the item at a given position in list 刪除
- Look at (retrieve) the item at a given position 檢索

心得：相對於第一章，較於理解，且在寫程式時有很多地方都可以使這章的內容實際使用後，也能讓程式看起來更整齊且容易理解。

單元3 鏈結串列

Pointers 指標 = 門牌 int *p

p=new int ; // 配置一個整數型態的空間

*p = 1; // 存放值到此空間

Dynamic Allocation of Arrays 動態配置陣列

ex.

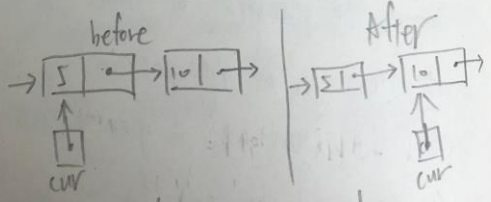
int arraySize = 50;

double *anArray = new double [arraySize];

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

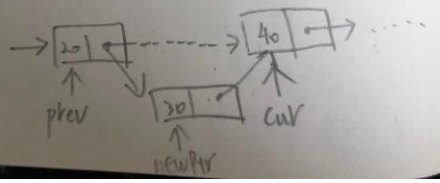
```
for(Node *cur =head; cur != NULL;
    cur = cur->next )
    cout << cur->item <<endl;
```

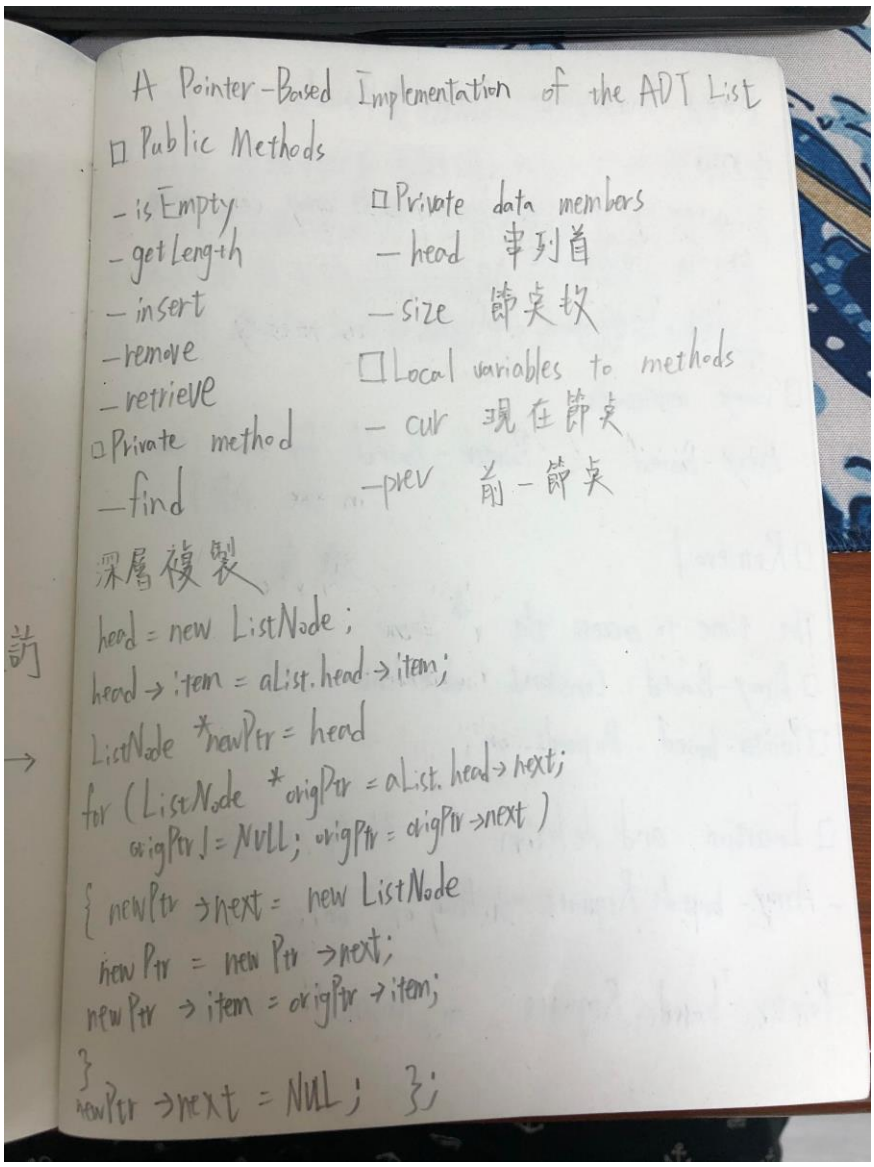
走訪



新增 □ To insert a node between two nodes

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





A Pointer-Based Implementation of the ADT List

□ Public Methods
- isEmpty
- getLength
- insert
- remove
- retrieve

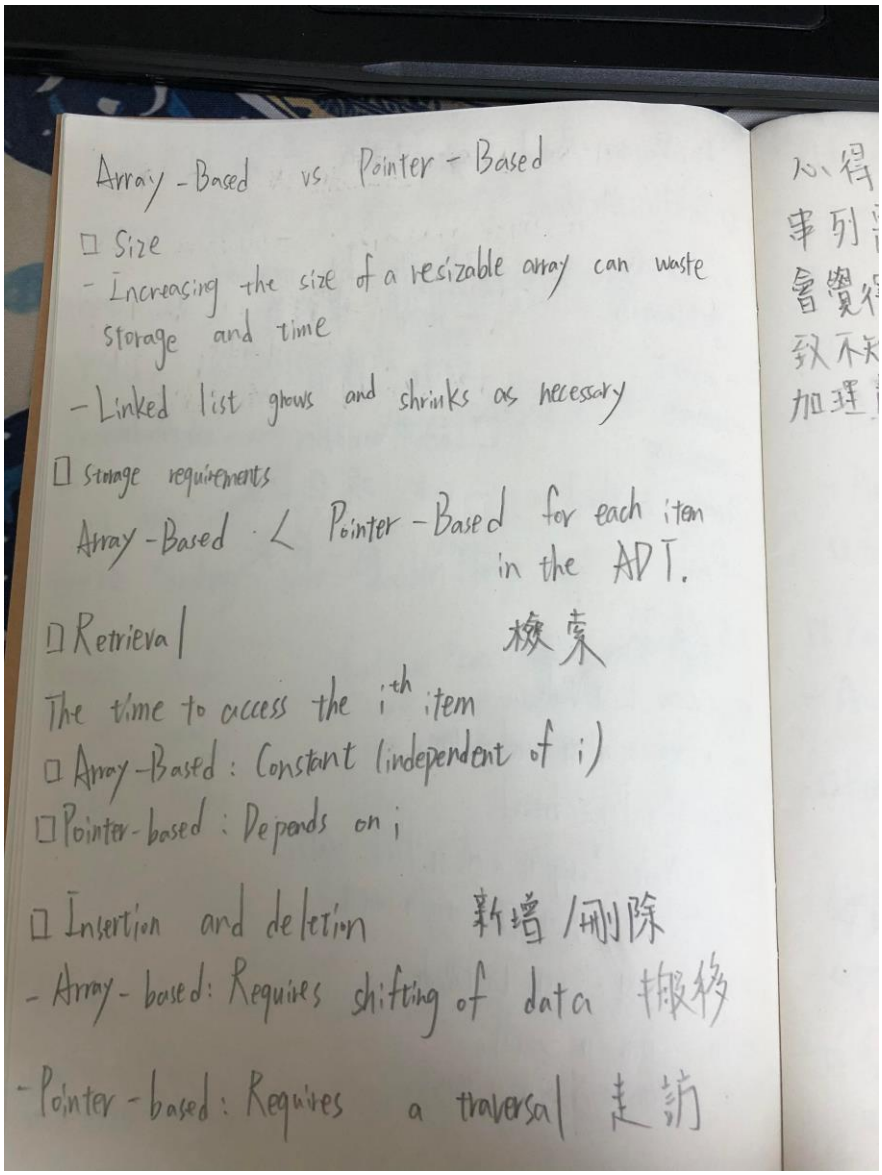
□ Private method
- find

□ Private data members
- head 串列首
- size 節点数

□ Local variables to methods
- cur 現在節点
- prev 前一節点

深層複製

```
head = new ListNode;
head->item = aList.head->item;
ListNode *newPtr = head
for (ListNode *origPtr = aList.head->next;
    origPtr != NULL; origPtr = origPtr->next )
{ newPtr->next = new ListNode
  newPtr = newPtr->next;
  newPtr->item = origPtr->item;
}
newPtr->next = NULL; };
```



Array-Based vs Pointer-Based

□ Size
- Increasing the size of a resizable array can waste storage and time
- Linked list grows and shrinks as necessary

□ Storage requirements
Array-Based < Pointer-Based for each item in the ADT.

□ Retrieval 檢索
The time to access the i^{th} item
□ Array-Based: Constant (independent of i)
□ Pointer-based: Depends on i

□ Insertion and deletion 新增/刪除
- Array-based: Requires shifting of data 搬移
- Pointer-based: Requires a traversal 走訪

心得：在大一計算机概论課程中，鏈結串列是讓我有印象深刻的之一，一開始學習到會覺得有点困難實做，常常會將節点亂接而導致不知道程式碼哪裡有錯，但後期自己也越加理解運作，反而成為我其中之一擅長的單元。

單元4 以遞迴解題

The Basics of Grammars

<number> = <digit><number> | <digit>

<digit> = 0|1|2|3|4|5|6|7|8|9

<addition> = <digit>+<addition> | <digit>

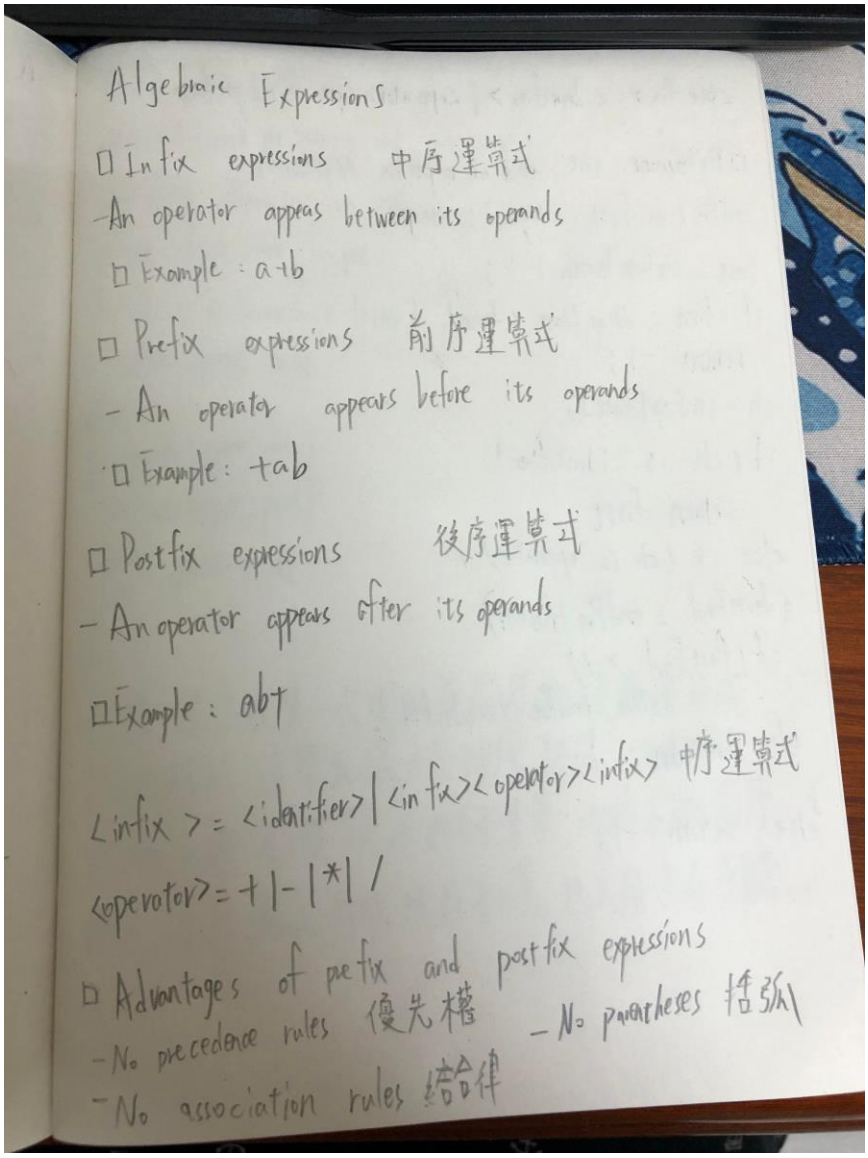
<digit> = 0|1|2|3|4|5|6|7|8|9

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

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

□ Recognition algorithm 回文

```
isPal (w: String) : boolean
if (w is empty or w is of length 1) 空字串 or 單一字
  return true;
else if (w's first and last are same letter)
  return isPal (w minus first and last)
else
  return false;
```



Algebraic Expressions

□ Infix expressions 中序運算式
- An operator appears between its operands
  □ Example : a+b

□ Prefix expressions 前序運算式
- An operator appears before its operands
  □ Example : +ab

□ Postfix expressions 後序運算式
- An operator appears after its operands

□ Example : ab+

<infix> = <identifier> | <infix><operator><infix> 中序運算式

<operator> = + | - | * | /

□ Advantages of prefix and postfix expressions
- No precedence rules 優先權
- No parentheses 括弧
- No association rules 結合律

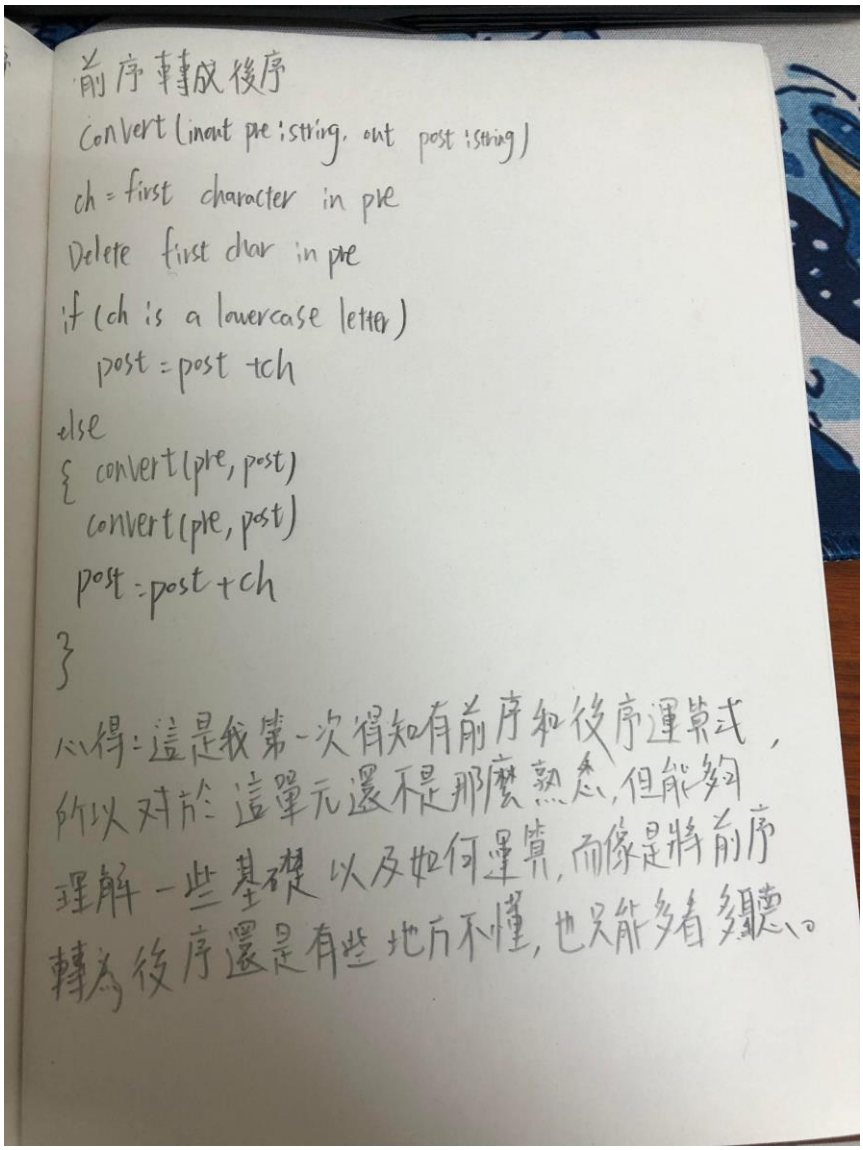
<prefix> = <identifier> | <operator> <prefix> <prefix> 前序

□ Determine the end of a prefix expression

endPre (in first : integer) : integer

last = strExp.length() - 1;
if (first < 0) or (last < first)
  return -1;
ch = strExp[first];
if (ch is identifier)
  return first
else if (ch is operator)
{ firstEnd = endPre (first+1)
 if (firstEnd > -1)
    return endPre (firstEnd+1);
else return -1;

else return -1;



前序轉成後序

```
convert (inout pre : string, out post : string)
ch = first character in pre
Delete first char in pre
if (ch is a lowercase letter)
   post = post + ch
else
{ convert (pre, post)
  convert (pre, post)
  post = post + ch
}
```

心得：這是我第一次得知有前序和後序運算式，所以对於這單元還不是那麼熟悉，但能夠理解一些基礎以及如何運算，而像是將前序轉為後序還是有些地方不懂，也只能多看幾遍。