單元1 遞迴.

遞迴 (recursion)

→把問題變越來越小並解決 (相同方法→做同一件事)

* fractal 碎形 * divide and conquer 分而擊之

① 把字串倒過來印

s.substr(size-1, 1); 印最後一個字元.

② 河內塔 (Towers of Hanoi)

→用最少的次數將一堆盤放到另一堆且只能小的在大的上

$\underline{2^n - 1}$ # = 幾次.

③ 費式數列 (Fibonacci Sequence).

$n_1 = 1$

$n_2 = 1$

$n_3 = n_2 + n_1 + 1 = 1+1+1 = 3$

$n_4 = n_3 + n_2 + 1 = 3+1+1 = 5$

$n_5 = n_4 + n_3 + 1 = 5+3+1 = 9$

n_k at least doubles every time.

不好

⇒ $n_k \geq 2^{\frac{k}{2}}$ 呼叫次數以指數成長

* 改善方法「用空間換時間」

ex. 用array將Ans存 利用查表,不再呼叫

④ n個選k個的組合數.

$C(n,k). = C(n-1, k-1) = C(n-1, k)$

必選地球 choose k-1 out of n-1

不選 〃 k n-1

Base case $c(k,k) = 1$ $C(n,k) = 0$ if $k > n$.

$c(n,0) = 1$

㊀

尾端遞迴 Tail recursion

```
void writeBackward (string s, int size){
    if (size > 0){
        cout << s.substr(size-1,1);
        writeBaokward (s, size-1);
    }//
}//
```

Recursive ×

↑
Tail recursion
改成 while
(最後一次, 什麼都不做)

```
void wB( string: s, int size)
    while ( size > 0 ) {
        cout << s.substr (size-1, 1);
        -- size;
    }//
}//
```

Iterative ○ 效率好

單元 2 抽象化. (Abstraction)

2-1 物件導向概念,

- 所有東西都是物件
  - data members (n) 屬性 (Attributes - characteristics)
  - methods or data functions (v) 運算 (Behaviors - operations)

- Encapsulation 封裝 - hides
  - Inheritance 繼承 - reused
  - Polymorphism 多型.

2-2 物件導向程式設計介紹.

- Purpose 目的
- Assumptions 假設
- Input
- Output

2-3 資料抽象化原理

- Modularity 模組化
  - Cohesion 高內聚 (模組做一件事, 互動多)
  - Coupling 低耦合 (關係不大, 傳遞參數低)

↓

功能性抽象化 (描述、實作)

2-5. 反轉序列資料 ⇒ 依值排序

- reverselist (in aList: List, out source: boolean)

```
for ( i=1 to aList getLength() -1 ){
    aList. retrieve (1, dataItem, success);     第一位
    aList. remove (1. success);
    aList. insert (aList. getLength() - i + 2), dataItem, success);
} // for
```

→ 先刪除後插入

2.   最後一位
retrieve (aList.getLength(), ‥)
insert (i , ‥)
remove (aList,getLength() , ‥)
↳不用加1, 因為還沒刪
所以總是6個

[1] [2] [3] [4] [5]
~~apples~~ ← milk ← bread ← butter ← juice
~~milk~~ ← bread ← butter ← juice ← apples
bread butter juice milk → apples
butter juice bread milk apples

⇒ juice, butter, bread, milk, apples

2-7 Practice.

1、改變約会目的

```
changeAppointmentPurpose ( in apptDate : Date,      10/8
in apptTime : Time , in purpose : string) : boolean      13:00
if ( is Appointment (apptDate, apptTime)) 存在
    cancel Appointment (    "    ); 取消
return make Appointment (    "    , purpose ); 建立新約会目的
```

-makeAppointment	新增
-cancelAppointment	取消
- isAppointment	是否有約
- checkAppointment	約会目的

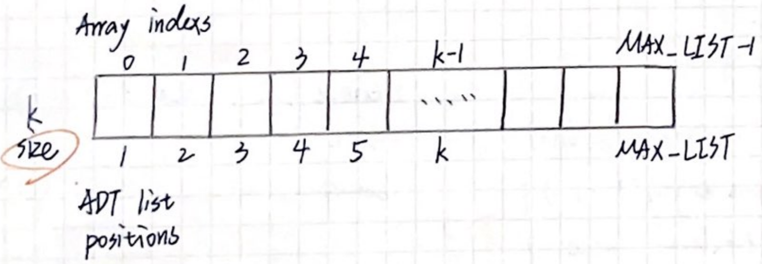
2、顯示指定日期所有約会.

```
displayAllAppointments (in apptDate : Date)
time = startofDay ;
while ( time < endofDay )
    if ( is Appointment ( apptDate , time ) 存在
    displayAppointment ( apptDate , time ); 有就印出來.
    time = time + halfHour ; ↗單位時間
```

- Private : only class instances

Protected : subclass instances

Public : any class instances



底層實作 : 陣列索引

高階描述 : 串列位置

insert : 後的位置都右移

remove : " 左移

Polynomial 多項式

1. degree () 最高項次的指數

2、coefficient (in power) 係數

3、changeCoefficient (in new coefficient, in power) 修改 Power 次項的係數

各項指數均為非負整數

ex. $p = 4x^5 + 7x^3 - x^2 + 9$

p.degree () = 5

P. coefficient (3) = 7

P、changeCoefficient (-3, 4) $P = 4x^5 - 3x^4 + 7x^3 - x^2 + 9$

P、 " (p.coefficient (0) , p.degree()) $P = 9x^5 - 3x^4 + 7x^3 - x^2 + 9$

$\overset{\shortparallel}{9}$ 最高次項

一、最高次項係數 1. display (p.coefficient (p.degree ())) ;

二. 將 x^3 項係數 +5 2. changeCoefficient (p.coefficient (3)+5 , 3) ;

三. P和q 相加成另一變數 r 3. for (i=0 ; i < p.degree() || i < q.degree ; i++)

r.changeCoefficient (p.coefficient(i)+q.coefficien(i), i)

實作 : $P: 4x^5 + 7x^3 - x^2 + 9$

	(1)	(2)	(3)	(4)	(5)	(6)
aList (List)	9	0	-1	7	0	4
	x^0	x^1	x^2	$\underline{x^3}$	x^4	x^5

degree : alist. getLength () -1 ;

coefficient : alist. retrieve (power +1, aCoefficient, success) { success return aCoefficient ; X return 0

change : alist、remove (power +1 , success)

if (success) a List. insert (power +1 , newCoefficient , success).

Namespaces 命名空間 ⇒ using.

std → 標準函式庫 ⇒ using namespace std.

Exception 例外狀況. try 設定保護範圍

catch 捕捉例外狀況

```
try {.. throw(type); ...}
 catch (type1) {
    statement(s); }
 catch( type 2 )
    statement(s);
 catch ( ... )
     statement(s)
```

跳脫範圍

⇒

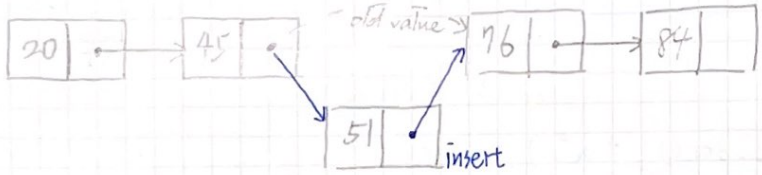
```
switch (type) {
  case (type1)
     statement(s);
     break;
  case ( type 2 )
     statement(s);
     break;
  default :
     statement(s);
 }.
```

單元 3 鏈結串列

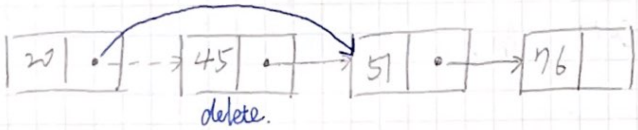
1. Linked Lists 指標

Array has a fixed size - 需移动資料 (陣列)
固定的
Linked List is able to grow in size as needed - 不需 (鏈結串列)

① 新增



② 删除



2. Pointers 指標 → 門牌號碼

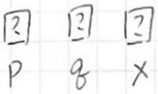
(int *)p ; (integer pointer)

p = new int ; (动態配置). 申請新房子 (delete p ; p = NULL;) → 清空.

p = &X → 房子x的門牌

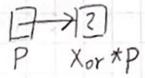
例.

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

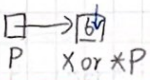


(b) p = & X

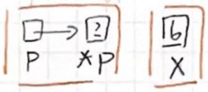
可對調



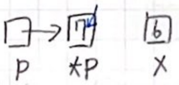
(c) *p = 6 ;



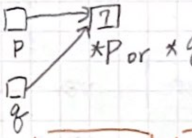
(d) p= new int ;



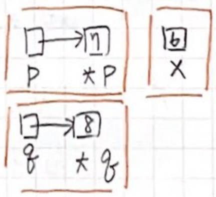
(e) *p = 7



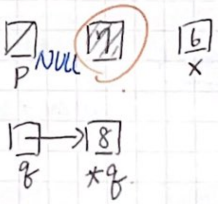
(f) q = p ;



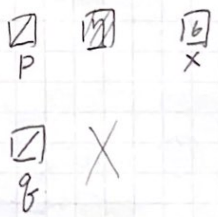
(g) q = new int;
*q = 8



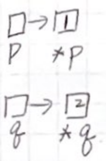
(h) p = NULL
X 錯誤
会造成 memory leak
(漏掉了).



(i) delete q
q = NULL

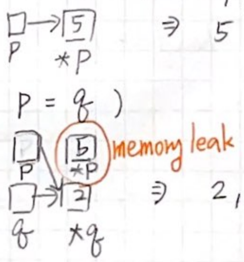


practice



⇒ (*p = *q + 3) ⇒ *p , *q ⇒ 5 , 2

(p = q) ⇒ 2 , 2



p=new int
*p = 7
delete p ; *p , *q
7? 2
不一定是7.
(dangling reference , illegal access).
p= NULL
memory leak ← q = NULL

3、动態（配置）陣列.

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

陣列名稱 = 指標

*需搬动

*存檔 save

```
fopen ( filename.c_str(), "a" );   "r"
fwrite ( &dA[i], sizeof(dA[i]), 1, fp );
```

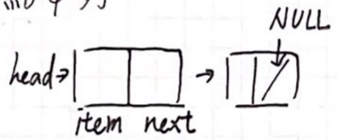
"a" → 寫資料到檔案.

"r" → 讀檔案

4. Linked List 鏈結串列

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

指標



宣告 `Node *p;`

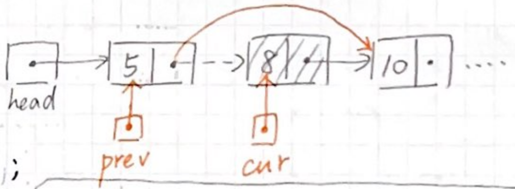
`p = new Node;`

[Delete]

```
prev → next = cur → next;
(prev → next = prev → next → next);
```

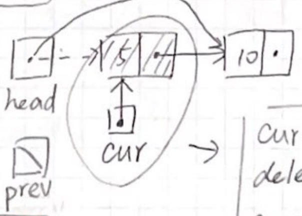
前一個

刪除位置



㊕ 刪除第一個 node

```
head = cur → next;
```



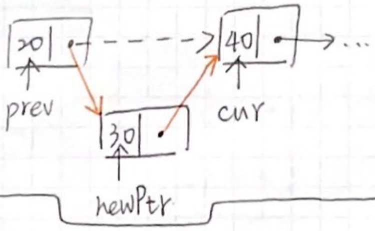
```
cur → next = NULL
delete cur;
cur = NULL
```

順序不可換

[insert]

```
newPtr → next = cur;
prev → next = newPtr;
```

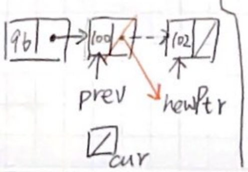
可對調



㊕ 加最後

```
newPtr→next = cur;
prev→next = newPtr;
```

可換

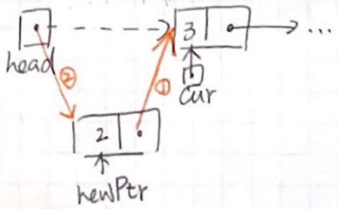


㊕ 加在第一個 node.

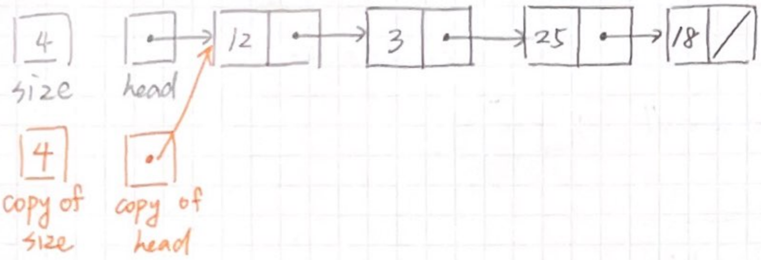
```
newPtr → next = head;
head = newPtr;
```

(cur)

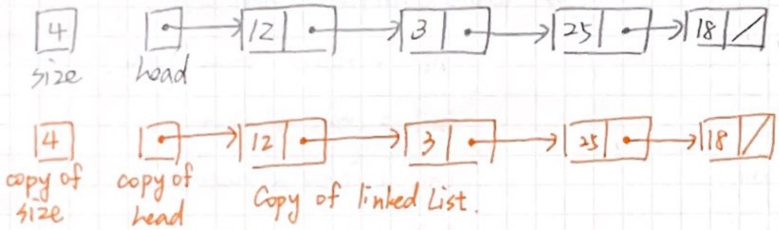
可對調 ☆



Shallow Copy 淺層複製

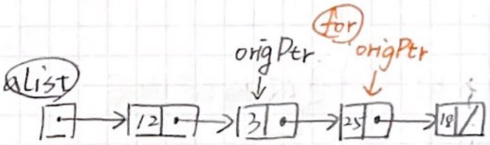


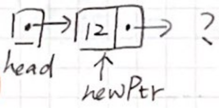
Deep Copy 深層複製

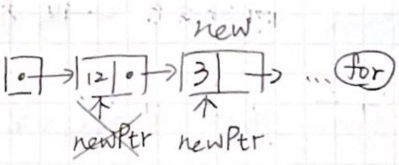


⇒ 對 copy 後的指標
做改變. 不影響原指標

```
         原始list
if ( aList . head == NULL )
    head = NULL;  // is empty.
else {.
    head = new ListNode.;
    head → item = aList . head → item ;  ⇒
    ListNode * newPtr = head  (不移动 head)
    for ( ListNode *origPtr = aList . head → next ;
          origPtr != NULL ; origPtr = origPtr → next ){
      newPtr → next = new ListNode ;
      newPtr = newPtr → next ;
      newPtr → item = origPtr → item;
    } // for
    newPtr → next = NULL;  將複制完的
                           最後設為NULL
} // else
```







存檔、只存資料不留指標

重建. 循序加入尾端 (tail).

```
ofstream outFile (fileName);    輸出檔

for (Node * cur = head ; cur != NULL ; cur = cur -> next)
    outFile << cur -> item << endl ;   只存資料
outFile.close().
```

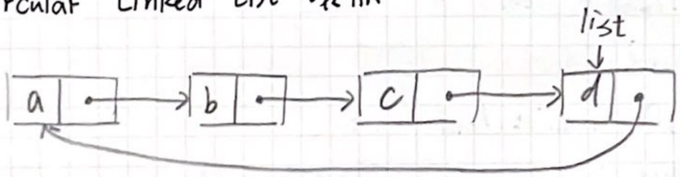
```
ifstream  inFile (fileName);
int  nextItem ;
if (inFile >> nextItem )      // 讀檔
    head = new Node;
    head -> item = nextItem;
    head -> next = NULL;
    tail = head;

while (inFile >> nextItem) {
    tail -> next = new Node;
    tail = tail -> next;
    tail -> item = nextItem;
    tail -> next = NULL;
} // while.
```

傳址 & : 更改指標內容.

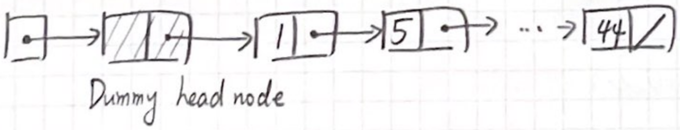
變型

* Circular Linked List 環狀



最後一個節点的下一個指標指向第一個節点

* Dummy Head Node (沒有例外狀況.)
空



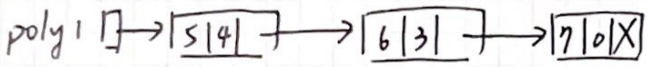
* Doubly Linked List
双向

```
struct Node {
    int item;
    Node * precede; //上一個
    Node * next;   // 下一個
}
```

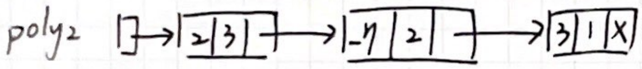
Circular + Dummy + Doubly
⇒ 三個不衝突

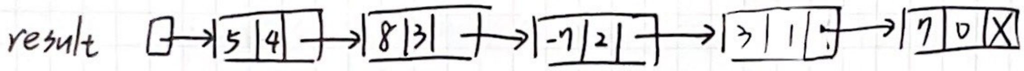
多項式加法.

$5x^4+6x^3+7$



$2x^3-7x^2+3x$





單元4 以遞迴解題

定義語言.

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

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

Grammer　　　　base case 終止條件.

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

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

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

* 算術運算式

operator 運算子 + - * /

operands 運算元

Infix expressions	a+b	中序運算式	((a+b)*C)/d
Prefix 〃	+ab	前序	/*+abcd
Postfix 〃	ab+	後序	ab+C*d/

完整括號.

中序運算式定義

<infix> = <identifier> | <infix><operator><infix>)

<operator> = +|-|*|/

<identifier> = a|b|c|...|z (變数名稱).

*中序轉前序.

((a+b)*c)

* + abc

* 中序轉後序

((a+b)*c)

ab+c*

前序 Grammer

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

<operator> = + | - | * | /

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

* 一個前序式後再接上非空字串一定不是前序式 ex. +*ab/cd-fg

Backtracking 僵局. 死巷 (退回).

→ 八皇后. 迷宮、数字迷宮.

* A search Problem. 兩點間路徑.

1. 抵達目的地
2. 無航班飛離城市
3. 重覆城市

* 数学歸納法

```
if (n is 0)                    特性.
    return 1
else
    return n * fact (n-1.
```

---

fact(0) = 0! = 1　最簡

fact(n) = n! = n * (n-1 * .. * 1　if n > 0　假設

fact(n+1) = (n+1) * fact(n)

= (n+1) * n! = (n+1) * n * ... * 1　歸納