遞迴 recursion.

功能.

階乘. 最大公因數. 搜尋. 費式數列. 組合數. 河內塔

Binary search

從最上層開始輸出

Ex 從 a 加到 b

100 ← 90

⇒ $\boxed{100} + (\cdots + (92 + (91 + 90)))$

最後一個加的

```
              100       90
int sum (int a, int b) {
 if (a > b)
   return (sum (a-1, b)) + a
 else  // a==b.
   return b ;
```

Ex 取餘數

gcd 1 (x, y) = x if y=0

= gcd 1 (x, y mod x) if y>x

= gcd 1 (y, x mod y) else

```
int gcd 1. (int x, int y)
 if (y==0) return x;
   if (y>0) return gcd1 (x, y%x)
    else return gcd1 (y, x%y)
```

效率

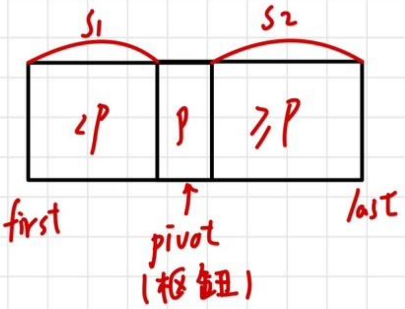
gcd 2 (x, y)=y if x mod y = 0

= gcd 2 (y, x mod y) else

```
int gcd 2 (int x, int y)
if !(x%y) return y;
else   return  gcd 2(y, x%y)
```

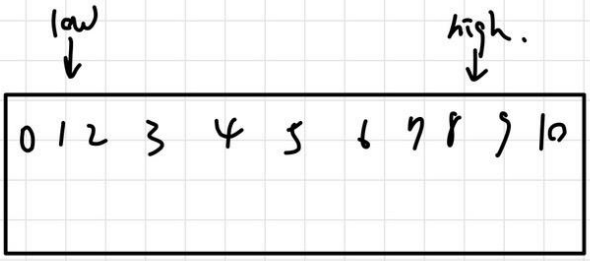
Ex：

找第k小的、



Ex：Reversing

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



河内塔.

```
              個数      起        終            輔助.
solve Towers (count, source, destination, spare)
                       A           C            B
if (count == 1)

else {
   solveTowers (count-1, source, spare, destination)
   solveTowers (1, source, destination, spare)
   solve Towers (count-1, spare, destination, source).
}
```

遞迴定義 . Base case rabbit(1), rabbit(0)

rabbit(n) = n、月 // if n is 0 or 1

= rabbit(n-1) + rabbit(n-2) if n>1

費氏數列

0.1.1.2.3.5.8.13.21.34 ...

How many times of recursive calls does it need? (遞迴次數)

rabbit#(n-1) + rabbit#(n-2) +1

(前2次需要的次數 + 自己一次)

線性費氏

```
if k=1
    return (k,0)

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

x^n

(a) 迴圈

(b) 遞迴

$x^0 = 1$

$x^n = x * x^{n-1}$, if $n>0$

(c) 二元遞迴

$x^0 = 1$

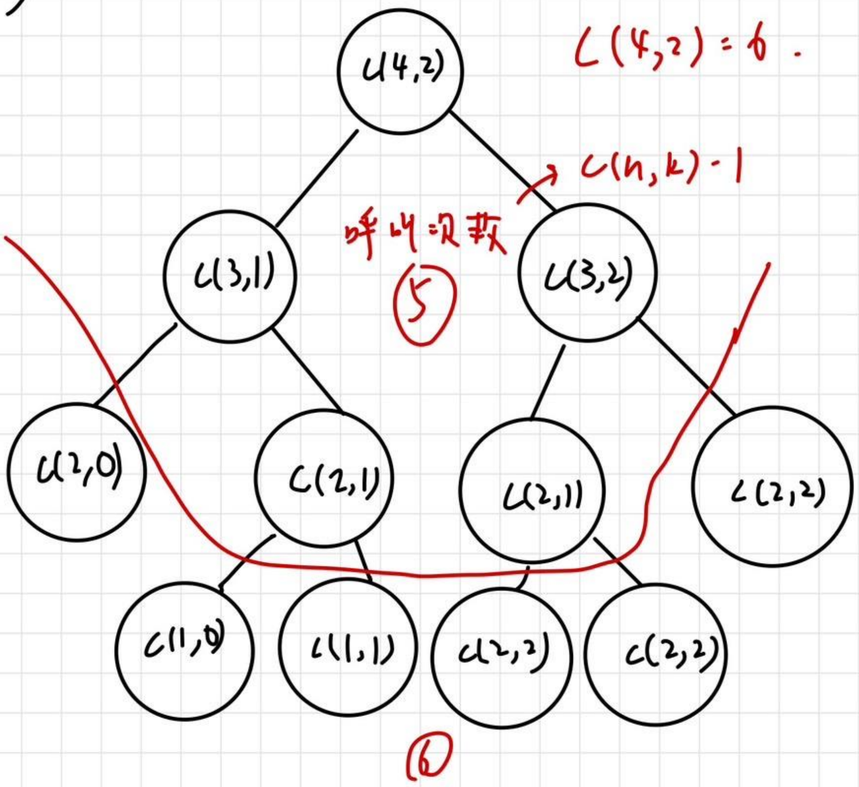
$x^n = (x^{n/2})^2$, if $n>0$ and n is even

$x^n = \underline{x} * (x^{n/2})^2$, if $n>0$ and n is odd

取 1 位

s.substr(size, 1) 字串最後一位開始.

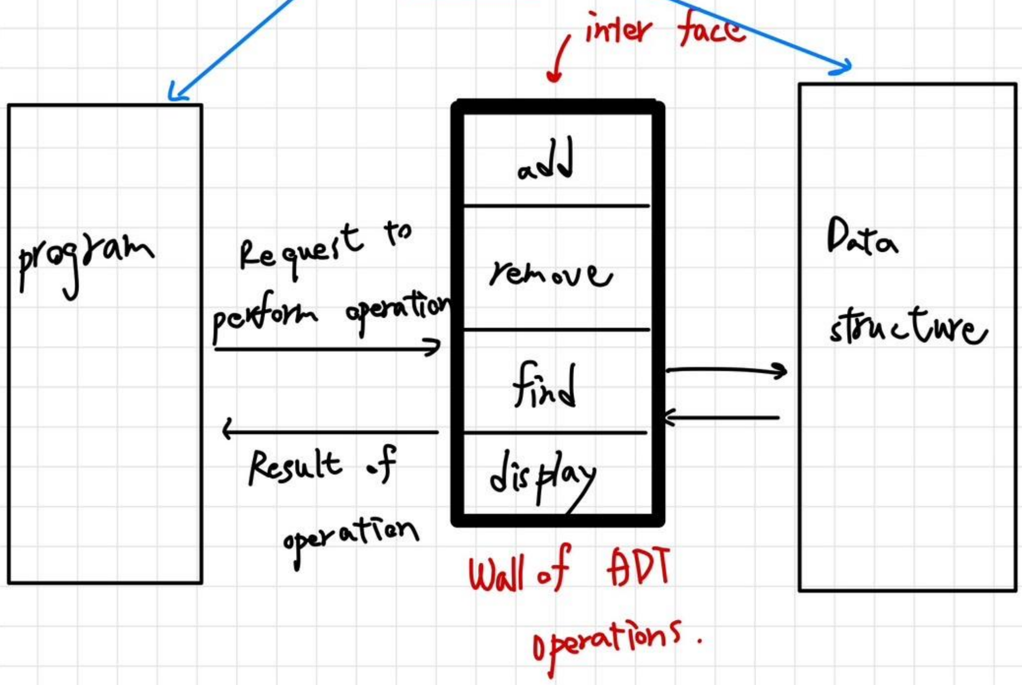
Binary trees



單元2. 抽象化.

Abstract Data Type (ADT)

分 < Specitification 描述
Inplementation 實作



The ADT List

```
                list        integer        ListItemType      boolean
replace ( in alist , in i , in newItem , out    success)
置換
    alist . remove (i, success);

    if (success)
        alist . insert(i, newitem, success)
```

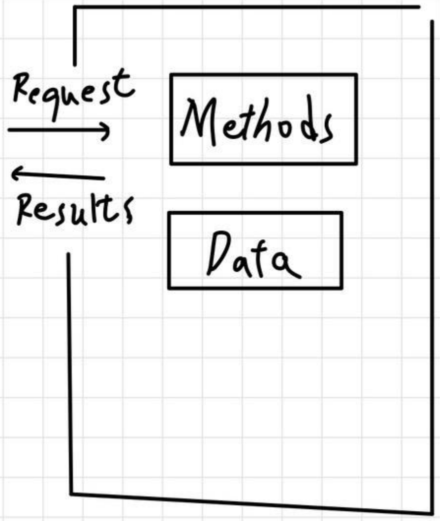
先刪除後插入

```
reverse(in alist , out success)
反轉.
  for (i = 1 to alist.getLength( ) -1 )
  { aList. retrieve (1, dataItem, success);

    alist . remove (1, success)

    alist . insert (alist.getLength( ) - i + 2, dataItem, success)
  }
```

```
reverse(in aList , out success)
反転.
 for (i = 1 to aList.getLength() -1 )
 { aList.retrieve(1, dataItem, success);

   aList.insert( i , dataItem, success)

   aList.remove (aList.getLength() , success)
 }
```

先插後刪

C++ classes.



封裝

ADT Polynomial 多項式

1. degree( ) 多項式最高次項的指數

2. coefficient (in power) Power 次項係數

3. changeCoefficient (in newcoefficient, in power)
   將 Power次項的係數改寫為 newCoefficient.

Using

1. 顯示最高次項係數

2. 將 x^3 項的係數 +5

3. 將 p和 q 相加成 r

```
1. display (p. coefficient (p.degree( ))) ;
2. p. changeCoefficient( p.coefficient( 3 )+5, 3 )
3. for(i = 0 ; i< p.degree( ) || i< q.degree( ), ++i)
       r. changeCoefficient(p.coefficient( i ) + q.coefficient(i),i)
```

```
degree {
    return aList . getLength( ) -1 ;
}
```

$4x^5 + 7x^3 - x^2 + 9$

(6)	(5)	(4)	(3)	(2)	(1)
4	0	7	-1	0	9

```
coefficient ( in power ) {
   aList . retrieve ( power +1, aCoefficient , success);
   if ( success )     return aCoefficient ;
   else               return 0 ;
}

change Coefficient {
     aList . remove ( power +1 , success);
     if ( success)
         aList . insert ( power +1 , newCoefficient, success);
```

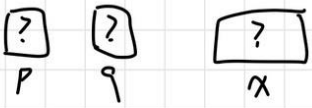
單元3. 鏈結串列.

delete p ;

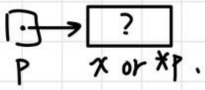
↳ p = NULL ;

習慣

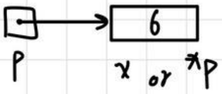
(a) int *p , *q ;
int x ;



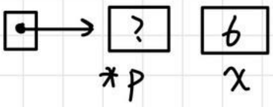
(b) p = &x ;



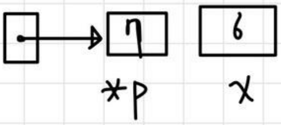
(c) *p = 6



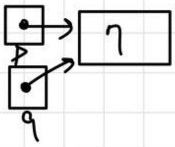
(d) p = new int ;



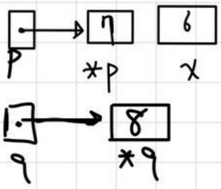
(e) *p = 7



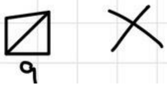
(f) q = p



(g) q = new int
*q = 8



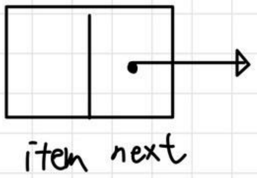
(h) delete q ;
q = NULL;



动態陣列 .

```
int arraysize = 50;
double* anArray = new double [arraysize];
```

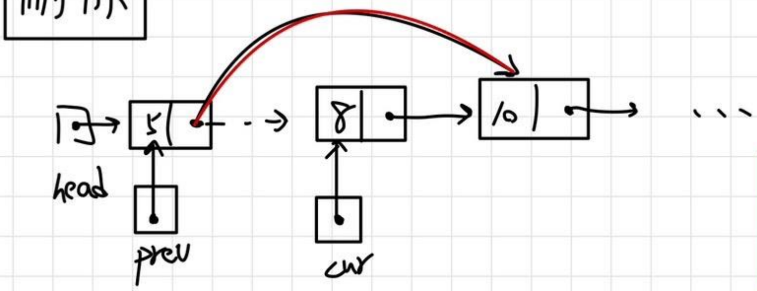
```
struct Node {
    int item;
    Node *next;

};
```



走訪

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

刪除



Delet first

head = cur → next

prev → next = cur → next .

or prev → next = prev → next → next .

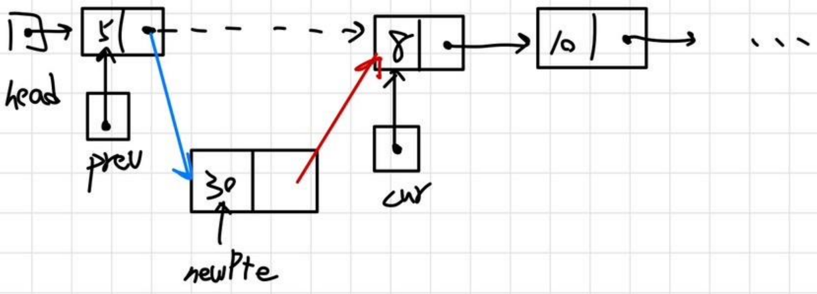
cur → next = NULL;

delet cur;

cur = NULL;

insert

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

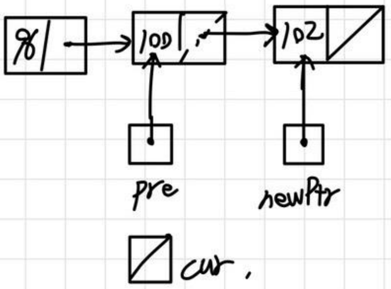


if first

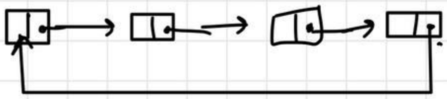
```
newPtr → next = Cur;
head = newPtr;
```

if end

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



- Circular Linked List



- Dummy head node

功能：Eliminate the special node

- Doubly Linked list

双向指標，初始值指自己.

```
{
  Node * precede
  Node * next
}
```

Write only data to a file, not pointers!

```
ofstream outFile
  outFile << cur→item << endl;   → 寫檔
outFile.close()
ifstream inFile
  infile >> next.item → 讀檔
```

單元4、遞迴.

定義語言.

x|y (x 或 y)

xy or x·y (緊接)

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

<number> = <digit><number> | <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 辨識演算法.

isId # 遞迴 終止條件：單一字元 / 遞迴呼叫：扣最後一个字元

isId suffix

Palindroms 迴文

ex 38 + 83 = 121

<pal> = empty string <ch> | a<pal>a | b<pal>b | … | Z<pal>Z|

<ch> = a|b|…|z|A|B|…|Z

$A^n B^n$

<Legal-word> = empty string | A <Legal-word> B

Ex:

<S> = <L> | <D><S><S>|

<L> = A|B

<D> = 1|2.

a. write all three character strings in the language.

1AA 1BB. 1AB. 1BA. 2AA. 2BB. 2AB. 2BA.

b. Write One string in this language that contain more than three characters

12AAB 100組

- 描述至少一個字母所組成的字串

第一個字母大寫，剩的小寫.

Algebraic Expression.

Infix	中序	ex: a+b
prefix	前序	ex: +ab
postfix	後序	ex: ab+

↓

優點: 優先权

No 結合率

括弧

中轉前 ((a+b)*c) → *+abc

Prefix

<prefix> : <identifier> | <operator> <prefix> <prefix>

Base case : one lower case letter is a prefix exp

recursive : <operator><prefix><prefix>

一前序式後面再接上非空字串一定非前序式 .

determine.

```
endPre ( in first : integer ) : integer.
    last = strExp.length() - 1
    if (first < 0 ) or ( last < first)     // base case
      return -1;
    ch = strExp [first] ;
    if ( ch is an identifier )
      return first ;
    else if (ch is an operator ). {
      firstEnd = endPre (first + 1 )     // 1st operand.
    if ( first End > -1 )
          return endPre (firstEnd +1);     // 2nd operand
    else    return -1;
   }
   else  return -1 ;
```

evaluate..

```
ch = first character in StrExp ;
Delete first ch in Exp ;
if ch is identifer
    return value .
else if ch is operator . named op {
    operand 1 = evaluate (strExp)
    operand 2 = evaluate (strExp)
    return operand1 op operand2 ; }
```

Backtracking 回溯

Eight Queens Problem

recursive .

Base case : 全走完.

recursive step : 填剩的欄位.

If can not place a queen in the current column ↗ need to backtrack

- 遞迴與 數学歸納法的關係

都用 Base case , solve smaller problems to derive a solution