單元1 遞迴

遞迴：把問題變小並解決，程式碼精簡且容易解釋（不求效率）

Practice 1-1.

two natural a, b , a > b , recursive function to compute the sum of all interger from a→b.

```
int sum (int a , int b) {
  if (a > b)
     return sum ( a-1 , b) + a ;
  else
     return b ;
}
```

Recursive version of kSmall

```
kSmall ( k: integer, anArray : ArrayType , first: integer , last : integer )
if ( k < pivotIndex - first + 1 )
   return kSmall ( k, anArray , first , pivotIndex - 1 )
else if ( k == pivotIndex - first + 1 )
   return p
else
   return kSmall ( k - ( pivotIndex - first + 1 ) , anArray, pivotIndex + 1 , last )
```

Towers of Hanoi :

```
solveTowers (count, source, destination, spare)
  if (count is 1)
     Move a disk from source to destination
else {
   solveTowers (count-1, source, spare, destination)
   solveTowers ( 1, source, destination, spare)
   solveTowers ( count-1, spare, destination, source)
}
```

Practice 1-3 : 算 x 的 n 次方

①

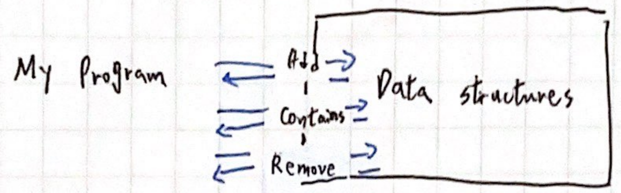
```
double power1 (double x, int n) {
   double result = 1;
   while (n>0) {
     result *= x;
     n--;
   }
   return result
}
```

②

```
double power2 (double x, int n) {
  if n(==0) return 1;
  else return x * power2 (x, n-1);
}
```

單元2：抽象化 Data Abstraction →抽象化

Cohesion (高內聚) v.s. Coupling (低耦合)



✿ predecesser 先行者 / successor 後繼者

{ 靜態 Attributes : data members
{ 动態 Behaviors : methods.

classes of objects
(called instances)

* 封裝 (Encapsulation)

繼承 (Inheritance) 重複使用程式碼

多型 (Polymorphism) 有選項

* ADT List Operations

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

常用

單元3. 鏈結串列

Pointers.

→ A pointer contains the location, or address in memory, of a memory cell.

Declaration of an integer pointer variable p.

```
int * p;
```

* Initially undefined, but not Null

```
p = & x;
```

&x = 房子 x 的門牌

The new operator `p = new int;` 申請一棟新房子

* std :: bad_alloc 沒有空房

```
delete p;
```

歸還房子

```
p = NULL  // safeguard
```

* 運用

☆ You can use the new operator to allocate an array dynamically.

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

陣列名稱 = 指標

☆ The size of a dynamically allocated array can be increased.

```
double * oldArray = anArray;
anArray = new double [3 * arraySize];
```

配置更大空間!

☆ 記得 delete! `delete [] oldArray;`

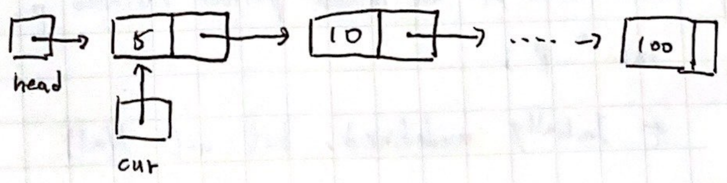
Pointer - Based Linked Lists

→ A node in a linked list is usually a struct.

→ If head = NULL, the link list is empty.

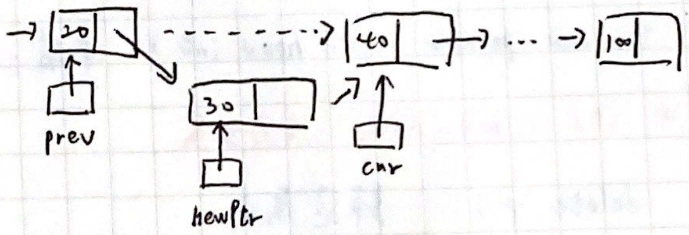
✲ Deleting the first node.

```
head = cur → next ;
cur → next = NULL ;
```



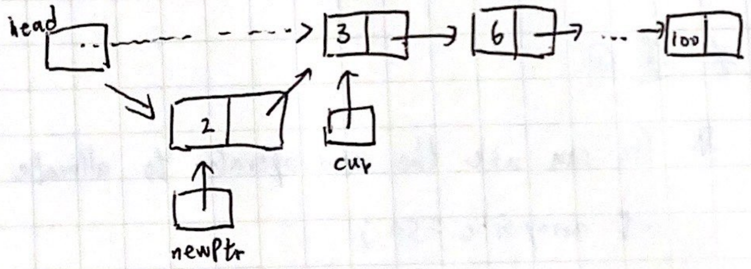
✲ Insert a node between two nodes

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



✲ Insert a node at the beginning of a linked list

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



單元4、以遞迴解題

If a C^{++} program is one long string of characters, the language of C^{++} programs is defined as:

C^{++} Programs = { strings w : w is a syntactically correct C^{++} program }

* The Basics of Grammars

1. $x \mid y$ means x or y

2. xy or $x \cdot y$ means x followed by y

* 辨識演算法

Algebraic Expressions

* Infix expressions

→ An operator appears between its operands. 中序運算式

$a+b$

* Prefix expressions

→ An operator appears before its operands. 前序運算式

$+ab$

* Postfix expressions

→ An operator appears after its operands. 後序運算式

$ab+$

To convert a fully parenthesized infix expression to a prefix form

→ Move each operator to the position marked by its corresponding open parenthesis

→ Remove the parentheses

Infix : $(a+b)*c$ Prefix : $*+abc$

Advantages of prefix and postfix expressions.

→ No precedence rules 優先權

→ No association rules 結合律

→ No parentheses 括弧

→ Simple grammars

→ Straight forward recognition and evaluation algorithms. 辨識 / 求解

Prefix Expressions

→ If E is a prefix expression

If Y is any nonempty string of nonblanks

Then E·Y connot be prefix.

一個前序式後面再接上非空字串一定不是前序式

Backtracking

→ A strategy for guessing at a solution a backing up when an impasse is reached.

→ Recursion and backtracking can be combined to solve problems.