

Subject: 資料結構 - 單元 1

No.:

Date:

遞迴: 精檢 - 能解相同且重複問題。

Box trace

```
void writeBackward (string s, int size)
```

```
{
```

```
  if (size > 0)
```

```
  {
```

```
    cout << s.substr (size-1, 1); → 輸出最後一個字元
```

```
    writeBackward (s, size-1); → 呼叫遞迴
```

```
  } // end if
```

```
  // do nothing
```

```
}
```

string **S** = "ABC" → output = "CBA"

int **size** = 3

求最大公因數

方法一:

```
int gcd1 (int x, int y) {
```

除到 0 結束

```
  if (y == 0) return x;
```

```
  else if (y > x) return gcd1 (x, y % x);
```

```
  else return gcd1 (y, x % y);
```

```
}
```

Subject:

方法二:

```

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

```

整除就結束

: $y > x$ 時和 $\text{gcd} 1$

次數相同

線性遞迴

1.

終止條件

2.

只有一路線

河內塔

個數 起點 終點 輔助點
 solveTowers (count, source, destination, spare)

if (count is 1)

else print \Rightarrow move a disk from source to destination

" (count-1, source, spare, destination)

" (1, source, destination, spare)

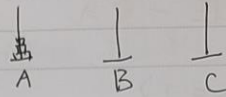
" (count-1, spare, destination, source)

Subject :

No. :

Date :/...../.....

河內塔 3 個 disk



(3, A, C, B)

(2, A, B, C)

(1, A, C, B)

(1, A, B, C)

(1, C, B, A)

(1, A, C, B)

(2, B, C, A)

(1, B, A, C)

(1, B, C, A)

(1, A, C, B)

Binary Recursion

兩個遞迴路線

費氏數列

Let n_k be the number of recursive calls

$$n_1 = 1$$

$$n_2 = 2$$

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

$$n_4 = n_3 + n_2$$

$$n_5 = n_4 + n_3$$

$$n_6 = n_5 + n_4$$

⋮

$$n_k \geq 2^{\frac{k}{2}}$$

次數以
指數成長

Subject :

No. :

Date :

Subject

使用線性 - 費氏

$LF(k)$

if ($k = 1$)

return ($k, 0$)

input: k

output: (F_k, F_{k-1})

else

(i, j) = $LF(k-1)$

return ($i+j, i$)


Subject: 單元 2 抽象化


No.:


Date:/...../.....

物件導向 2-1 概念

三大特徵: Encapsulation 封裝 · Inheritance 繼承
polymorphism 多型

 : can hide inner details

 : can reused

 : can find correct method

2-2 物導設計概念

1. 目的 2. 假設 3. 輸入 4. 輸出

2-3 抽象化原理

模組化: 有系統 · 容易修

高內聚 Cohesion: 每個模組做一件事, 互動多。

低耦合 Coupling: 傳遞參數少, 好維護。

No. :

Date :/...../.....

Subject :

2-4 資料形態

predecessor 先行者
successor 後繼者

置換 : 先 remove 再 insert
retrieve : 只讀資料

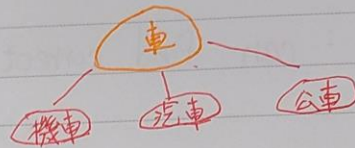
ex.

父類別 \Rightarrow sphere

子類別 \Rightarrow colorsphere

父類別 : base class

子類別 : derived class



protected : 讓原 private 的 data 繼承可使用

overriding 覆載 \Rightarrow 參數列相同

\Rightarrow 都是增加程式彈性

overloading 多載 \Rightarrow 參數列不同

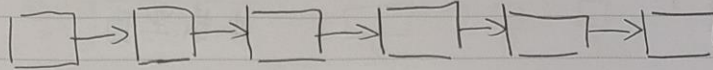
單元 3 鏈結串列

Subject:

No.:

Date:/...../.....

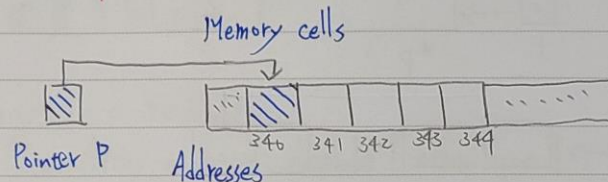
ex.



Array 陣列: 需移動資料

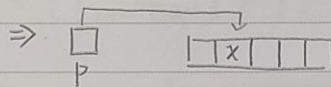
Linked list 鏈結串列: 不需移動資料

指標: $\text{int } *p \Rightarrow$ 門牌號碼



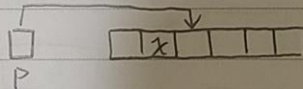
$\text{int } x;$

$p = \&x;$



$\&x =$ 房子 x 的門牌

$p = \text{new int};$



tip:

if the operator `new` cannot allocate memory, it throws the exception `std::bad_alloc` (in the `<new>` header)

↳ 要不到記憶時可使用

`delete p;`

歸還

`p = NULL;`

清空

\Rightarrow 以免誤用

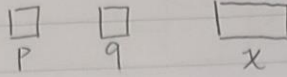
No. :

Date :/...../.....

Subject :

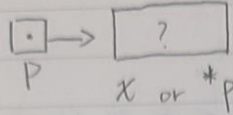
ex.

int *p, *q ;
int x ;



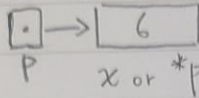
申請空白

p = &x ;

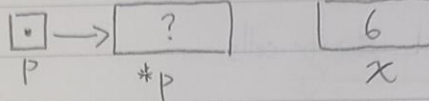


抄寫

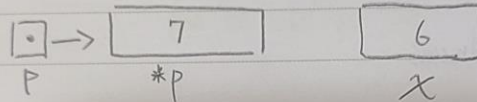
*p = 6 ;



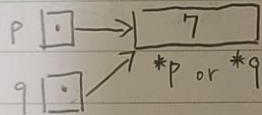
p = new int ;



*p = 7 ;

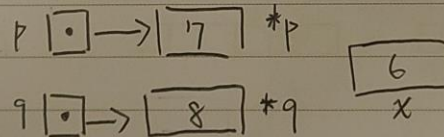


q = p ;

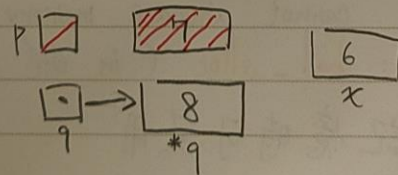


q = new int ;

*q = 8 ;



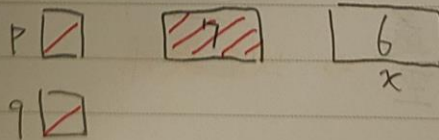
p = NULL ;



可能造成
memory leak

delete q ;

q = NULL ;



Subject :

No. :

Date :

```
p = new int ;  
*p = 7 ;  
cout << *p << " " << *q << endl ; // 7 2
```

```
delete p ;  
cout << *p << " " << *q << endl ;  
p = NULL ;  
q = NULL ;
```

dangling reference

memory leak

動態(配置)陣列

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

```
anArray[z] = *(anArray + z) ;
```

陣列名稱 = 指標

```
double *oldArray = anArray ;  
anArray = new double [ 3 * arraySize ] ;  
150
```

配置更大空間

```
for ( ... )
```

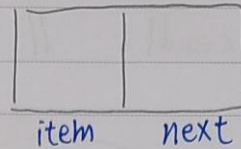
```
an ... [ ] = old ... [ ] ; // copy
```

```
delete [ ] oldArray ;
```

刪 150 個

A node in a linked list is usually a struct

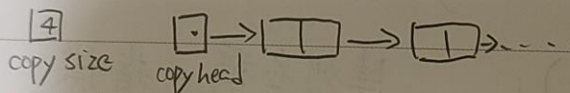
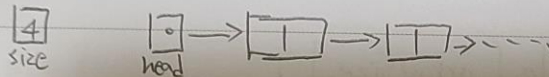
```
struct Node {
    int item;
    Node *next;
};
```



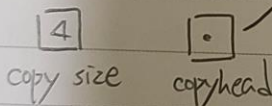
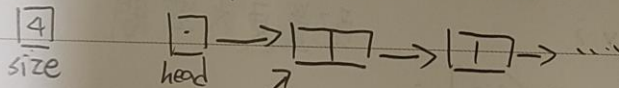
```
Node *p;      空門牌
p = new Node; 空房
```

深層複製: Copies size, head, linked list

ex.



淺層複製: Copies size, head



Subject: 深層複製建構子

No.:

Date:

```
#include <cstdlib> // for NULL
#include <new> // for bad_alloc
#include "ListP.h" // header file
using namespace std;
```

```
List::List (const List& alist) 複製建構
                               : size(alist.size) 節點數
```

```
{
    if (alist.head == NULL)
        head = NULL;
    else
    {
        head = new ListNode;
        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;
        } // end for
        newPtr->next = NULL;
    }
}
```

Subject :

No. :

Date :/...../.....

(find)

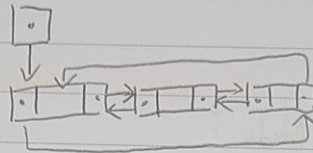
陣列 : size 固定、省空間 (單 item)、常數時間 (快)
需搬動資料

(find)

鏈結串列 : size 有彈性、費空間 (多存位置)、線性時間 (慢)

Double Linked list : 雙向, 多加一個指標
但要多維護一個指標

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



單元 4 以迴圈解題

Subject:

No.:

Date:

中序運算式 Infix expression

ex.

$a + b$, $a + b \times c \div d$

operator 運算子

operand 運算元

前序 Prefix

ex.

$+ a b$, $\div x + a b c d$

檢查 左括弧

後序 Postfix

ex.

$a b +$, $a b + c \times d \div$

檢查 右括弧

前序和後序的優點:

No precedence rules 不用優先權

No association rules 不用結合率

No parentheses 不用括弧

Simple grammars

Straightforward recognition and evaluation algorithms