電機三甲_10928156_蔡宜呈

CH 1

1-1 遞迴如同鏡子，互相映射且本質不變

ex. 碎形 fractal

binary search - divide and conquer 分而治之

1-2

遞迴是將問題縮減，將10人問題變成9人問題，8人…1人

ex. reverse a string of character → process one word at a time
→ process n times

1-3

寫出來的程式碼，結果雖然相同，但過程的差距卻也十分重要

1-4

思考問題
↓
簡化問題
↓
終止條件
↓
確認終止

1-5

判斷遞迴寫法的好壞 → 以遞迴次數來衡量 → 越少越好

☆ 觀察特殊案例

綜合上述2點做判斷

1-6

遞迴 → 參數傳遞來控制資料量

遞迴不一定比較有效率

1-7

通常若是處理一半資料的遞迴，效率較高

ex. k-th small

find a pivot ←

compare the two parts → find which part of kth small is in

1-7

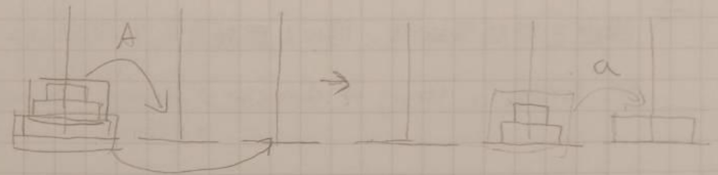
linear Recursion

→ 即便有很多種遞迴，也只擇一遞迴

先將問題縮至 base case 再陸續做下去

1-8

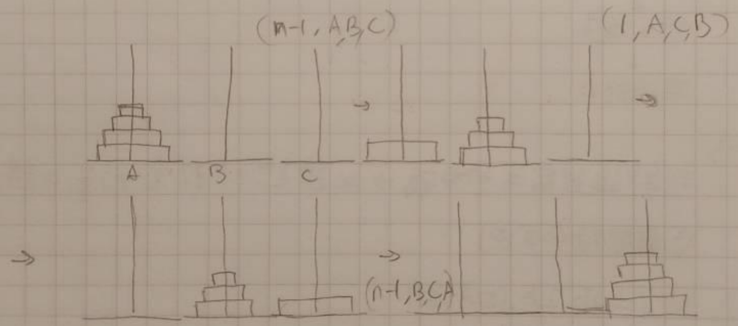
河內塔



A's step is same as a's step
(recursion)

1-9

將最大一片跟其餘分成2部分，可將程式分成四步驟



1-10 Binary - recursion

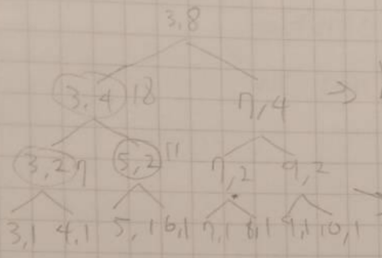
要印5個虛線則要印4個虛線，要印4個的虛線，則要印4個……
…… 可用遞迴

draw3
└draw2
 └draw1
  └drawtick 1 —
 └drawtick 2 — —
 └draw1
  └drawtick 1 —
└draw3 — — —
 └drawtick3

draw3 → draw2 → draw1
└→ draw2
 └draw1

1-11 另一種 binary recursion

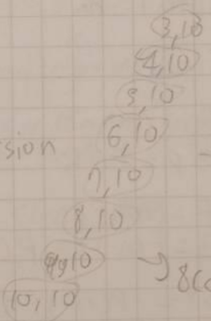
分一半



→ binary recursion

→ 15 calls

→ n-1 additions



→ linear recursion

→ 8 calls

→ n-1 additions

不一定誰比較有效率

1-12

rabbit 1th 1
2th 1
3th 2
4th 3
5th 5
6th 8

1, 1, 2, 3, 5, 8 ⋯

→ rabbit n = rabbit (n-1) + rabbit (n-2)

as same as fibonacci sequence

多餘的遞迴 → $n = (n-1) + (n-2)$

1-13

用 binary-recursion 不是不行，就是要配合 array 來儲存

用 linear-recursion 較優

fibonacci sequence

binary-recursion 呼叫次數以指數成長

linear-recursion 呼叫次數以線性成長

1-14

① iterative function

② recursive function - linear → $x^n = x \times x^{n-1}$

much efficiency → ③ recursive function - binary → $x^n = (x^{n/2})^2$ n is even

$= x \cdot (x^{\frac{n}{2}})^2$ n is odd

ex. 9^{32}

① 32 multiplication ① x

② 32 " ② 32 recursive calls → ② binary 不一定沒效率

③ 7 " ③ 7 "

1-15

total = P(n)

F(n) (end with float)

B(n) (end with a board)

$\Rightarrow P = F + B$

$F = P(n-1)$

$B = F(n-1) = P(n-2)$

$\Rightarrow P = P(n-1) + P(n-2)$

⇒ 簡化問題

1-10 Acker

$Acker(m,n) = n+1$ if $m=0$

$= Acker(m-1,1)$ $n=0$

$= Acker(m-1, Acker(m,n-1))$ otherwise

→ 爆很快

→ 適合用空間來換時間

用 array 記

1-17 $C(n,k) = C(n-1,k-1) + C(n-1,k)$

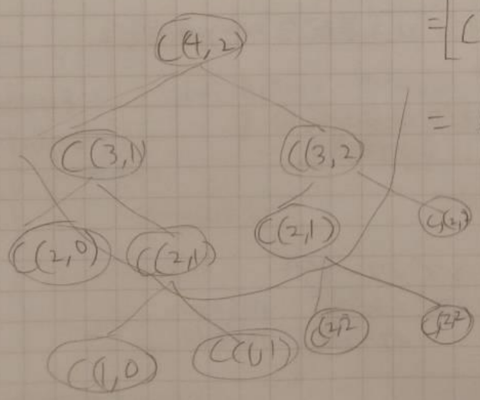
必選A　　不選A

$\rightarrow C(4,2) = C(3,1) + C(3,2)$

Binary trees

$= [C(2,0) + C(2,1)] + [C(2,1) + C(2,2)]$

$= [C(2,0) + C(1,0) + C(1,1)] + [C(2,2) + C(1,0) + C(1,1)]$



$= 6$　　|leaf nodes| − |internal node| = 1

→ leaf nodes → recursive call to base

internal node → non-base

C(n,k) = leaf node

recursive call = |internal node|

= |leaf node| − 1

1-18

Tail recursion → (while) iterative

sometimes more efficiently

We can change the location of the code to satisfied it

CH2

2-1

Classes of object (called instances)

Attributes : data members
Behaviors : methods

→ Principle of Object-Orented Programming

封裝 Encapsulation → hide inner details

繼承 Inheritance → reused

多型 Polymorphism

2-2

Operation Contracts

— document and limit the use of a method

the action - module assume, availabe data, effects on data

Key Issue in Programming

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

2-3

Abstract Data type - motive

Modularity

Cohesion → single task per module

Coupling → low desired dependence module

Functional abstraction
→ Detail it's behave

2-3 abstract data type - concept concept

build a wall between two module to manage it's process and data

manage it layerly

obstract data type - goals

through the specification exchange data with the programm

2-4

ADT list

predecessor 先行者

succesor 後繼者

insert, remove, retrieve

The wall between ~~ADT~~ displaylist and the implementation of ADT list

2-5

ADT sorted list

The ADT sorted list

operate item by value instead of position

2-6 Design ADT

add some appropriate ADT

- data required
- operation required

2-7

operation? → avoid the conflict → ex same type of data or existed data

detect the inform → respond

2-8

Implementing ADT

program and Data structure must comunicate through the wall of operations

C++ classes

→ an object is an instance of a class
→ define a new data type
→ contain method and data member
⇒ public and private

Constructor
- set data member to initial value

Destructor
Destroy an object when the object lifetime end

2-9

deprived class or sub class will inherit ~~the~~ the data members or method in class, or superclass

Class A : public B

A is B B isn't A B : 父 A : 子

PS 可以不只继承1個

no 2-10 & 2-11

2-12

Overloading ⇒ will find the function most fit ~~is~~ match

overriding ⇒ write in the same class with the same name

2-13

An array-based ADT list

turn translate into remove retreive insert and compare the implement

2-14 ADT - Polynomial

先宣告 → 再設計

最高次項的指數 → degree()
power次的係數 → coefficient (in power)
將power次的係數改成 newCoefficient → change(coefficient
(in newcoefficient, in Power)

2-15
記到 list 的 polynomial 的次項和 list 差1

2-16
Namespace → use (::) to access out side the namespace
declare "using" to directly use it

ex: using namespace xxxxx.

xxxxx::0000

a set (
try block →
catch block
)

```
try
{
  statement(s);   -> when something may go wrong
}
catch(ExceptionClass, identifier)
{
  statement(s);   -> process the "error"
}
```

```
try{.... throw(type); ...}
catch(type1)
{
  statement(s);
}
catch(type2)
{ statement(s);
}
```

```
void myMethod(int x) throw(Except)
{
  if(...)
    throw Except("Except: ...");
}
```

2-16

Define ADT fully before making any decisions

Hide the ADT by Define in C++ class

CH3

3-1

linked list → 

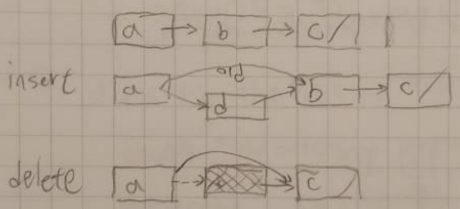
static and arrange orderly (there must be head and End)

ADT list

→ array → Data must be shifted during inserts and deletes

linked list → Data doesn't required shifting during inserts and deletes

ex.



a, b, c, d may be a large amount of data, that take times to transport, through changing the link (pointer) of a, b, c, d, we may able to do it in a more efficient way.

3-2

pointer - contains location or address in memory of memorycell

ev. (int *) p; ← the star

initially undefined, but Not Null

→ static allocation

→

*p ⇒ represents the memory cell

place the address of a variable into a pointer variable

⇒ The address-of operator "&"

p = &x; → give p the address of x & the value of x

"new" operator

p = new int; → dynamic allocation ⇒ change as x change

3-2

p = new int;

→ if "operator" new can't allocate memory it throws exception std::bad_alloc

→ it means no address and space for operator new

when u ~~want~~ don't want to use pointer anymore

u can use delete p; → just delete the address but the inform won't disappear

than p = Null; → much safer way

↳ every thing about p will vanish

we may accidently use it's address again

3-3

① int *p
int x

[?] [?]
p x

② p = &x

[ ] → [ ]
p x or *p

一定要先 → ②再③

③ *p = 6

⇒ [ ] → [6]
p x or *p

④ p = new int;

[ ] → [ ] [6]
p *p x

⑤ *p = 7

[ ] → [7]
p *p

memory leak (must avoid)

→ p = Null [ ] [7]
p ↑ still there

delete p; → [ ]
p

p = Null → [ ]
p

3-3

after using pointer remember to clear it to prevent memory leak

```
{ delete p;
  P=NULL;     -> write together
```

3-4

new operator to allocate an array

```
int size = 50
double *array = new double[size]
```

-> array name is a pointer to its first element

∴ array[2] ≡ *(array+2)

-> increase the size of the array

```
double *old array = array
array = new double [3*size]
```

↳ if want to transfer the data in old array into new "array" we have to insert 50 times

-> delete[ ] oldarray;

↑ need

3-5

有些程式處理檔案只能藉由檔案編號交給 OS 來處理

有"new"時就要有delete

try catch → 容易有錯誤時用

3-6

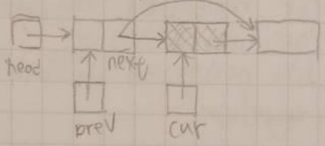
a node in link list is usually a struct

```
struct Node
{ int item
  node *next;     -> dynamic allocated
};                   Node = *P
                     P = new Node;
```

```
head = new Node
delete head;
head = NULL;
```

3-7

Delete Node



prev → next = cur → next ;

⇒ 將 cur 的 Next 移至 prev 的 next

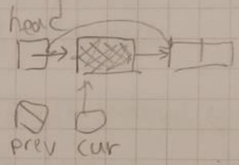
or

prev → next = prev → next → next ;

Delete first Node ⇒ prev = NULL

⇓

head = cur → next



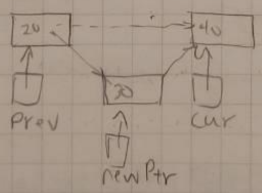
must do 順序 不能亂

⇒ cur → next = NULL

delete cur ;

cur = NULL ;

add a new Node



newPtr → next = cur

prev → next = newPtr

add a new Node at head

newPtr → next = head

head = newPtr

add a new Node at the tail

newPtr → next = cur

Prev → next = newPtr

Node *prev, *cur

for (prev=NULL, cur=head;
(cur != NULL) && (newValue
> cur->item; prev=cur,
cur=cur->next

3-8

new & delete -> dynamic allocation
Node each pointer link to next node

3-9

In dynamic memory allocation the constructor and destructor both two have to write by urself

Destructor has a name same as constructor ~~but~~ have a dash ~

3-8

Delete the Node a Specific position

```
Node *prev, *cur;
if (head != NULL)
{ for (Prev=NULL, cur=head; (cur != NULL) && (new Value > cur->item);
    prev=cur, cur=(cur->next);

if (prev=NULL)
    head = cur->next
else prev->next = cur->next;
cur->next = NULL;
delete cur
cur = NULL;
```

Insert a Node to Specific position

```
for (prev = NULL, cur = head; (cur != NULL) && (newValue > cur->item);
    prev = cur, cur = (cur->next);
if (prev == NULL)
{
  newPtr->next = head;
  head = newPtr;
}
else
{ newPtr->next = cur;
  prev->next = newPtr;
}
```

3-9

Public methods	Private data members
- isEmpty	- head 串列首
- getlength	- size 節点數
- insert	local variables to methods
- remove	- cur
- retrieve	- prev
Private method	
- find	

3-10

```
List::List(const List & aList) : size(aList.size)
{
 if (aList.head = NULL)
    head = NULL;
 else
 { 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;
 }
}
```

3-11

insert

```
ListNode *newPtr = new ListNode;
Size = newLength;                 ← 產生新節点
newPtr -> item = newItem;
if (index == 1)
    newPtr -> next = head; ← 將新節點插入串列中
    head = newPtr;
else
   ListNode *prev = find(index-1); ← 先找到前一個節点
   newPtr -> next = prev -> next;        再改變指標
   prev -> next = newPtr
```

remove

```
--size;
if (index == 1)
     cur = head;
     head = head -> next
else {
   ListNode *prev = find(index -1); ← 先找到前一個節点
   cur = prev -> next;
   prev -> next = cur -> next
}
cur -> next = NULL;
delete cur; ⟶ 歸還空間給系統
cur = NULL;
```

3-12

Compare Array-Based, Pointer-Based

Memory storage Array 較省

Retriveral time , Array 較少

insert and delete 各有優缺

3-13

save & Restore Link list

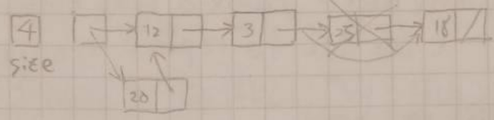
ofstream

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

ifstream

```
head = new Node
  head->item = nextItem
  head->next = NULL
  tail = head → 第一個節點
  while( inFile >> nextItem)
  { tail->next = new Node
    tail = tail->next
    tail->item = nextItem
    tail->next = NULL;
```

3-14



0	12	1
1	3	2
2	25	3
3	18	-1
4		5
5		6
6		-1

$\boxed{0}$ head

$\boxed{4}$ free

0	12	1
1	3	3
2	25	4
3	18	-1
4		5
5		6
6		-1

$\boxed{0}$ head

$\boxed{2}$ free

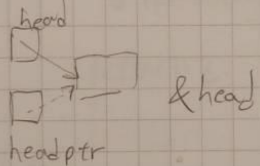
0	12	1
1	3	3
2	20	0
3	18	-1
4		5
5		6
6		-1

$\boxed{2}$ head

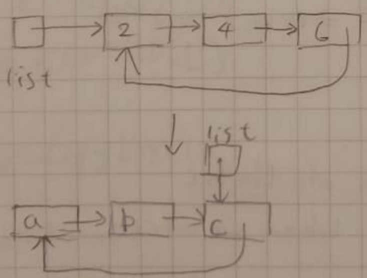
$\boxed{4}$ free

↗ 不需搬動整行，只要將指標指定的內容改變就好

3-15



3-16



→ circular

→ No Node contains NULL

```
→ Node *first = list->next      ← point to first node
  Node *cur = first              ← start at first node
  do { show(cur->item);
     cur = cur->next;            → move to next node
  } while (cur != first)         → list
```

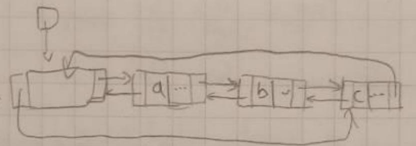
3-16

dummy head node → create a blank node at first of the list to replace the utility of "head"

→ prev = head

*next 下一個

*precede 上一個



Dummy head Node →

delete → (cur → precede) → next = cur → next

(cur → next) → precede = cur → precede

insert

newPtr → next = cur;

newPtr → precede = cur → precede;

cur → precede = newPtr

newPtr → precede → next = newPtr

3-17

List
Find
Replace
Insert

} operation of the list when we write an link list

3-18

```
Node *addpoly(Node*x, Node*y)
{ Node*z= NULL, *w =NULL
if ① x or y is empty
else {
  ② a dummy head node
  do{
    ③ create a new node

    ④ copy p & c, then find next

  } while (x!=NULL)&&(y!=NULL);
  ⑤ remaining x or y }
return z;
}
```

```
① if(x==NULL)
     z =copylist(y);
  else if(y==NULL)

     z=copyList(x)

②
  w=new Node

  z =w;

③
  z->next=new Node
  z=z->next
```

```
④ If(x->p>=y->p)
  { z->p=x->p;
  if(x->p==y->p)
  {z->c=x->c+y->c;
   y=y->next
  } else z->c=x->c;
   x=x->next
  }Else{
   z->p=y->p
    z->c=y->c
    y=y->next
```

1-1

Defining language

the symbols , + - × ÷ · |

ex. Grammer: ex

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

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

<addition> = <number> + <addition> | number

C++ Identifier

⇒ C++ Ids = {w : w is a legal C++ identifier}

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

<letter> = 0|1|2|..|a|b|...|z

```
isId(in w:string):boolean
  if (w is of length 1)
    if (w is a letter)
      return true;
    else
      return false;
  else if (the last character of w is a letter or a digit)
    return isId(w minus its last character)
  else
    return false;
```

4-2

palindromes: 回文 ex. Hannah

<pal> = empty string | <ch> |

a<pal>a | b<pal>b | ....

. <S> = <L> | <D><S><S> → L or DSS

<L> = A|B

<D> = 1|2

pos of 3character = 1AA 1AB 1BA 1BB

2AA 2BA 2AB 2BB

4-3

Algerbric Expression

→ Infix expressions

ex. A + B → ((a+b)*c)/d

Prefix expressins

ex. +AB → /*+abcd

Postfix

ex. AB+ → ab+c*d/

((a+b)*c) → 運算子對應左括號 → *+abc

Infix prefix

((a+b)*c) —— 右 —— ab+c*

infix postfix

4-3 advantage of prefix & postfix

→ No precedence rules

→ No association rules

→ No parantheses

4-4

grammer of prefix

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

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

4-5

~~back~~

try and error then backtrack

is a good way for some question

and it's also important to avoid the

impossible ways to decrease the

possibility to wrong end

4-6

list out all the possible way ~~that~~

If there is a way that conform as our need

than stop, otherwise backtrack to the last

place we choose.

4-7

Prove by induction on $|E|$

Basis $|E|=1$: E is a single letter → don't begin with operator

→ not prefix

hypothesis → $1 \le |E| < n$ if E is prefix, then EY is NOT prefix

$|E| = n$; $E = op\ P_1\ P_2$, both P_1 and P_2 are Prefix

$|P_1| < n$ $|P_2| < n$

assume EY is Prefix

assume $EY = op\ W_1\ W_2$ where W_1 are prefix

$\Rightarrow W_1 = P_1$

$\Rightarrow$ contradiction

tower of HANOI

Basis $N=1$ moves$(1) = 2^1 - 1 = 1$

hypothesis $1 < N < k$ moves$(N) = 2^N - 1$

Inductive step $N=k$: moves$(k) = 2 * \text{moves}(k-1) + 1$

$= 2 * (2^{k-1} - 1) + 1$

$= 2^k - 1$