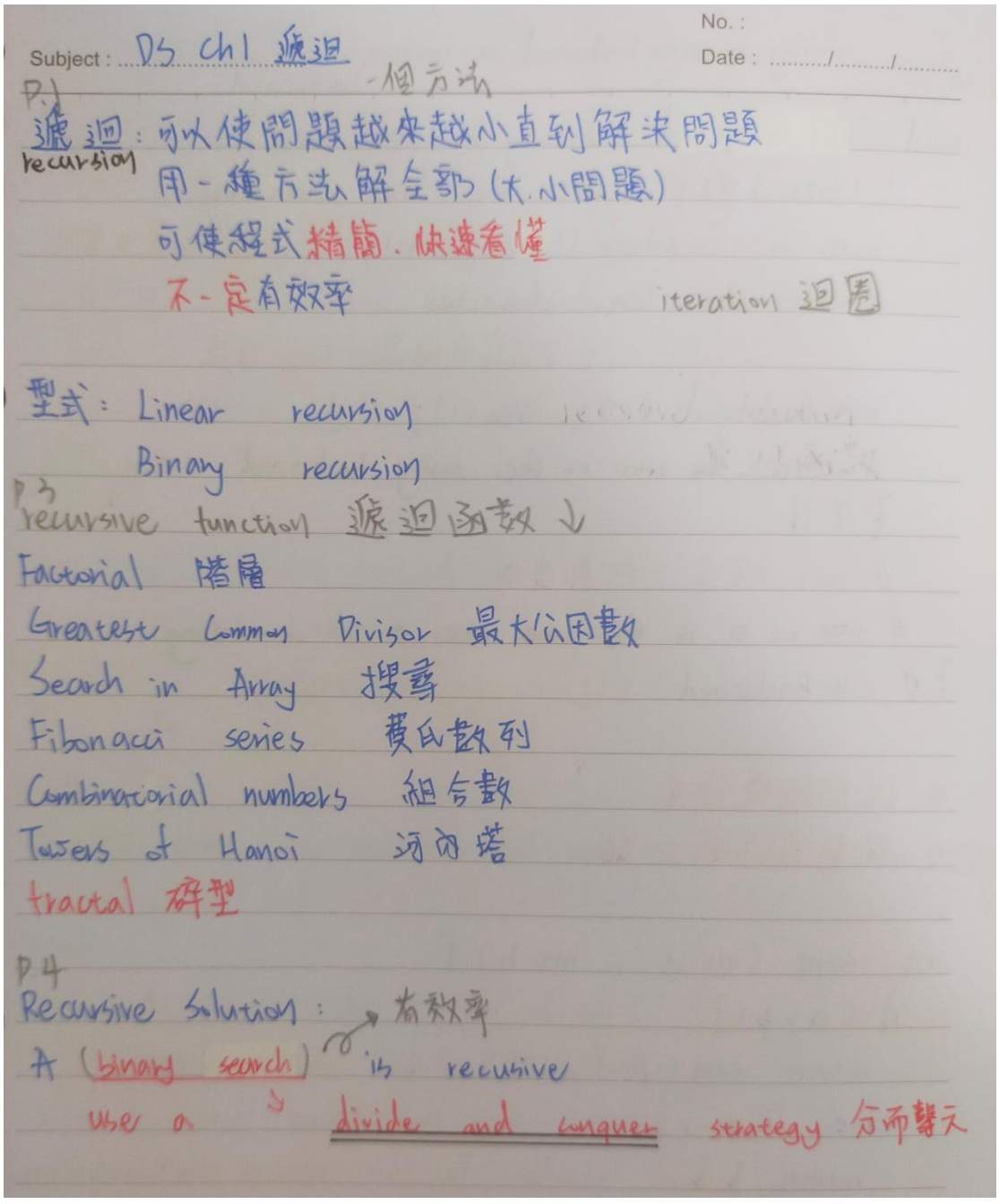
CH1



Subject : DS ch1 遞迴 No. :
Date : ...../...../.....

P.1
遞迴 recursion：可以使問題越來越小直到解決問題
用一種方法解全部(大.小問題)
可使程式精簡.快速看懂
不一定有效率 iteration 迴圈

型式：Linear recursion
Binary recursion

P.3
recursive function 遞迴函數
Factorial 階層
Greatest Common Divisor 最大公因數
Search in Array 搜尋
Fibonacci series 費氏數列
Combinatorial numbers 組合數
Towers of Hanoi 河內塔
fractal 碎型

P.4
Recursive Solution：有效率
A (binary search) is recusive
use a divide and conquer strategy 分而擊之

P.14 Subject: writing a string backward in reverse backward Date: ..../..../....

```
void writeBackward ( string s , int size ) {
  if (size > 0) {
    cout << s.substr ( size-1 , 1 ); // 輸出位置先要!!
    // write the last character
                    抓字串的某一部分出來
    writeBackward ( s , size-1 );
    // write the rest of the string backward
  } // if
  // cout 放這：印原字串
  // size == 0 is the base case -> do something
} // writeBackward
```

* 讓問題變簡單
* 設計需有停止點.

P.19

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

```
if (a == b) {
  return a;
else
  return sum ( a , b+1 )+b;
```

1. 遞迴定義
2. 問題簡化
3. 終止條件
4. 保證終止

Subject : greatest common divisor Date : ...../...../.....

P.22

試圖讓數值越來越小 base case 整除就完蛋。

• 假設 x=9 , y=6 哪一作法較有效率？

```
int gcd1 (int x , int y) {
  if (y == 0) return x;
  else if (y > x) return gcd1 (x , y%x);
  else return gcd1 (y , x%y);
} // gcd1

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

P.23

⇒ gcd2 saves one recursive call when x divides y (when x≥y)

special case : x=6 y=9 當 x<y

P.28

finding the largest item in an Array

if (array only one item) // 終止條件.

else if (array more than one item)

maxArray (left half of anArray) and / 求這

maxArray (right half of anArray)

⇒ 沒效率 !!! 可以解但沒必要！

Subject : finding the k^{th} smallest item in array Date : ...../...../.....

P.30

<p <p p >p >p

(pivot index

selecting a pivot item in the array

可只找其中一邊

ex：左3筆 找第10小 看右就好.

ksmall (k, anArray, first, last) k=4

20 7 25 19 2 16 12 48 39

12 7 25 19 2 16 20 ~~48~~ ~~39~~

12 7 2 // 16 19 25 20

2 7 12 16 19 25 20 ⇒ 也可以排序

kSmall (k, anArray, f, L)

if k < pivot - first + 1 kSmall (k, anArray, first, pivot -1);

if k > pivot - first + 1 kSmall (k, anArray, pivot+1, last);

if k = pivot - first + 1 return p;

P.1 Linear Recursion

☆ Test for base cases 檢查終止條件

☆ Recur once 只擇一遞迴

P.5

Reverse an Array 反轉次序

if low < high then

Swap anArray[low] and anArray[high]

ReverseArray (anArray, low+1, high-1);

⇒ 程式好看. 不見得效率高

1-08

Subject : Towers of Hanoi

No. :

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

個 起 終 輔

Algorithm towers (numDisks, source, dest, auxiliary)

```
if ( numDisks == 1 )
    Move a disk directly from source to dest
else {
    towers ( numDisks-1 , source, auxiliary, dest)
    towers ( 1 , source, dest, auxiliary)
    towers ( numDisks-1, auxiliary, dest, source)
} // else
```

可把遞迴呼叫畫成樹狀圖

呼叫次數和搬動次數不同

Binary Recursion

: occurs whenever there are two recursive calls for each non-base case

```
void drawTicks (int tickLength)
 if (tickLength > 0) {
    drawTicks ( tickLength -1 );
    drawOneTicks (          );
    drawTicks ( tickLength -1 );
  } // if
} // tickLength
```

Subject : 程式碼 再寫過

No. :

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

```
void drawRuler (int nInches, int majorlength)
    drawOneTick (majorlength, 0);
    for (int i=1; i< nInches; i++) {
        drawTicks (majorlength -1);
        drawOneTick (majorlength, 1);
    } // for
} // drawRuler
void drawTicks (int ticklength) {
    if (ticklength > 0){
        drawTicks (ticklength - 1);
        drawOneTick (ticklength, -1);
        drawTicks (ticklength -1);
    } //if
} // drawTicks

void drawOneTick (int ticklength, int ticklabel) {
    for (int i=0; i< ticklength; i++)
        cout << "-";
    if (ticklabel >=0)
        cout << " " << ticklabell;
    cout << endl;
} // drawOneTick
```

刻度+數值

R有刻度

幾個刻度

1-17

ubject : .....

No. :

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

// Binary recursion

```
int SumB ( int a , int n ) {
  // assume n = b - a + 1
  if (n == 1)  return a;
  return sumB (a, n/2) + sumB ( a+n/2 , n - n/2 )
} // SumB
```

遞迴呼叫：Binary 的 SumB > linear SumA

加法：相同

Multiplying Rabbits

How many pairs of rabbits are alive in month n ?

組成：

Base case : rabbit (1) . rabbit (1)

Recursive : if n is 1 or 2 return (n) = 1

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

Fibonacci : rabbit(1) rabbit(2) rabbit (3) → 費氏數列.

1 1 2 3 5 8 13 ...

```
int rabbit (int n) {
  if (n <= 2)  return 1;
  else return rabbit(n-1) + rabbit(n-2);
} // rabbit
```

Subject : Fibonacci Sequence Date : ..../..../....

1, 1, 2, 3, 5, 8, 13, 21, 34 ...

Better Fibonacci use linear recursion

algorithm linearFibonacci (k)

Input: A nonnegative integer k

Output: Pair of Fibonacci numbers (F_k, F_{k-1})

if k=1 return (k, 0) // base case k=1 (F_1, F_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}

呼叫次數以線性成長

1-14 Practice 1-13

3 ways to compute x^n for nonnegative integer n :

(1) write an interative function power1

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

(2) Write a recursive function power 2

$x^0 = 1$ $\quad x^n = x \cdot x^{n-1}$ if $n > 0$

呼叫次數 n
乘法次數 n-1

```
double power2 (double x, int n) {
    if (n == 0) return 1; or n == 1 return x
    else return x * power (x, n-1)
} // power 2
```

x x x x
4 3 2 1

(3) Write another recursive function power 3 二分法遞迴

$x_0 = 1$ $\quad x^n = (x^{n/2})^2$ if $n > 0$ and n is even

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

```
double power3 (double x, int n) {
  if (n == 0) return 1;
  else {
    double halfpower = power3 (x, n/2);
    if (n%2 == 0) return halfpower * halfpower;
    else return x * halfpower * halfpower;
  } // else
} // power 3
```

compare 9^{32} 9^{19} multiplications

in power2. 3 recursive calls

CH2

Subject : Ch2 資料抽象化 Data abseration Date : ...../...../.....

principals of object-oriented progamming.

=> enables us to build classes of objects (called instance)

=> a class combine

Attributes of objects of a single type 屬性

- typically data.
- called data members

Behaviors (operations) 運算

- typically operate on the data
- called methods or member functions

=> three characteristics

Encapsulation 封裝

- objects combine data and operations
- Hides inner details

Inheritance 繼承

- classes can inherit properties from other classes
- existing classes can be reused

Polymorphism 多型

- objects can determine appropriate operations at execution time

ADT

Subject : abstract data types : motives

No. :

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

Modularity 模組化 (修改)

- 將程式切成小塊. 系統好管理. 易讀易寫易找 bug
- Isolates errors

cohesion 高內聚 : 盡可能讓一個模組只做一件事

coupling 低耦合 : 少量參數傳遞

Functional abstration

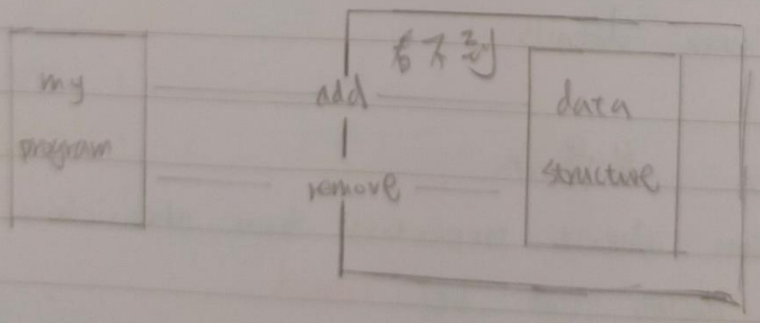
purpuse and use 講清楚

描述. 寫作, 也是 別人不用看懂

specification implementation ⇒ 兩者是分開來的

要告別人看法

ex. 製冰機. 外表使用 and 內部運作.



Subject : ADT list operations

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

- create an empty list createlist() 建構
- destory a list destorylist() 解構
- whether a list is empty isEmpty() 是否為空
- the number of items in a list getLength() 計算個數
- insert insert() 插入
- delete remove() 刪除
- look at retrieve() 檢索

應用: reverse the entire list

```
reverse list (in aList:List, out source:boolean){   先刪除
    for (i=1 to aList.getlength()-1){                後插入
        aList.retrieve (1, dataItem, success);
        aList.remove (1, success);
        aList.insert (aList.getlength()-i+2, dataItem, success)
    } // for
```

the ADT sortedlist 依值排序 need value

Determin the dates of holiday

```
listHoliday (in year : integer)
  date = date of the first day of year
  while (date is before the first day of year+1){
    if (date is a holiday) write( date, "is a holiday")
    date = date of the next day
  } // while
```

Subject : implementing ADTs

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

以 class 來封裝 可將 members / functions 分為 public / private

inheritance in C++ : protect : subclass instances inherits any of the publicly methods or data members of a base classes or superclass

an instance of a derived class can invoke public methods of the base classes

class 可用一樣的名稱，傳進來的參數可設不一樣

ex : int add (int num)

int add (long num)

CH3

Subject : CH3 Linked Lists Date : ...../...../...

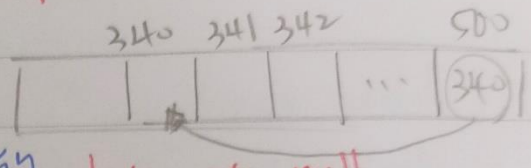
鏈結串列：提供彈性

* 指標 Pointers 類似於門牌號碼

array : linked list in ADT

⇒ 需要搬移動 : 不需要搬動.

Pointers (int *p;



還沒有房子：空的 but not null

int x;

p = &x;

&x = 房子x的門牌.

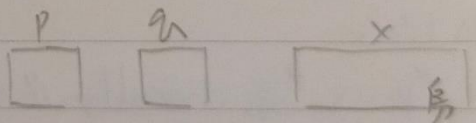
p = new int;

有可能沒有空房.

有 new 完後不要要記得 delete 不然可能沒空間給予

刪完要再加上 p = NULL 要徹底遺忘;

缺的話可能還會住甲到已規還的空間



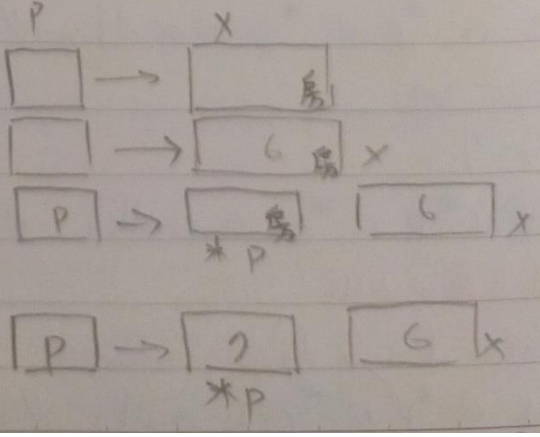
(a) int *p, *q;
int x;

(b) p = &x) 不能對待

(c) *p = 6

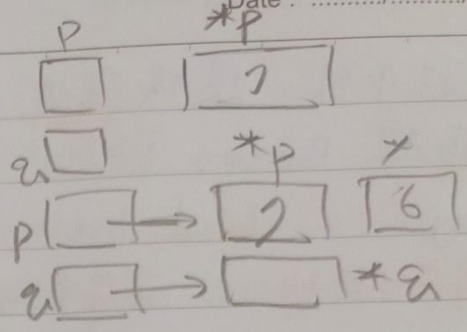
(d) p = new int

(e) *p = ?



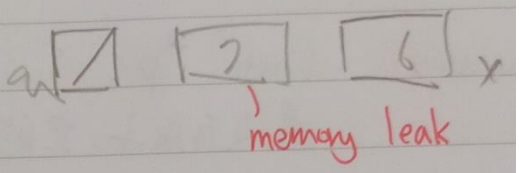
Subject : ..... Date : ...../...../...

(f) `q = p;`



(g)
```
q = new int;
*q = p;
```

(h)
```
delete q;
q = NULL;
```



*memory leak

```
int *p, *q;
p = new int;
*p = 1;
q = new int;
*q = 2;
*p = *q + 3;
cout << *p << *q << endl; // 52
p = q   // memory leak
cout << *p << *q << endl; // 22
p = new int
*p = 7;
cout << *p << *q << endl; // 7 2
```

習慣上接 = NULL

```
delete p; → dangling reference illegal access
cout << *p << *q << endl; // 有可能 run time error
p = NULL
q = NULL
```

Subject : Dynamic Allocation of Arrays 動態陣列 Date : ____/____/____

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

陣列名稱 ≡ 指標　　anArray[2] ≡ *(anArray+2)

Ex. 配置更大空間.

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

Save/Copy a File

```
typedef struct student {
  char sid[SID_LEN];
  int score;
} studentType;
```

```
void saveFile (FILE*, studentType[], int);
```

```
void saveFile (FILE *fp, studentType dA[], int no){
  for (int i=0; i<no; i++){
    fwrite (&dA[i], sizeof(dA[i]), 1, fp);
    cout << dA[i].sid << "," << dA[i].score << endl;
  } // for
  fclose (fp);
} // saveFile
```

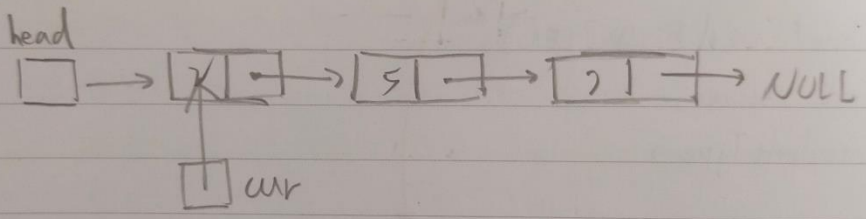
```
FILE * outfile = NULL;
string fileName = "DSample1.data";
studentType alls[SR_NUM] = {("110x113", 60), ~};
outfile = fopen(fileName.c_str(), "a");
if (outfile != NULL) saveFile(outfile, alls, SR_NUM);
return 0;
} // main()
```

若事先不知道學生筆數?

```
studentType *buts;
studentNo = ftell(infile);
buts = new studentType[studentNo];
```

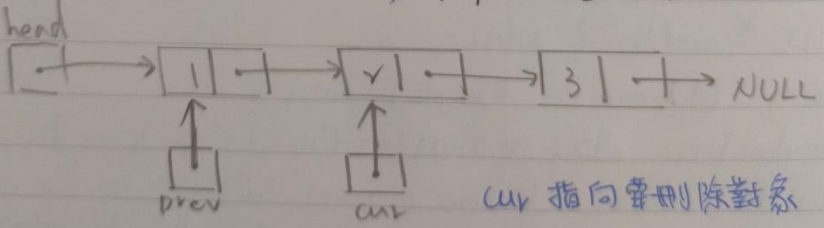
Display 走訪.

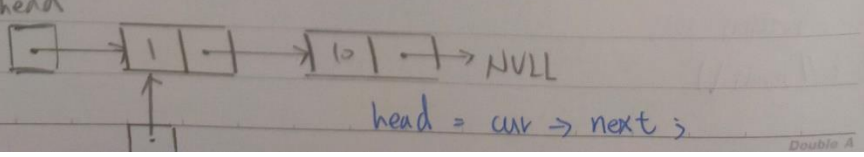
```
for (Node *cur = head; cur != NULL; cur = cur -> next) {
    cout << cur -> item << endl;
} // for
```



Delete 刪除

```
prev -> next = cur -> next; // 需要兩個變數
```






```
head = cur -> next;
```

avoiding dangling reference

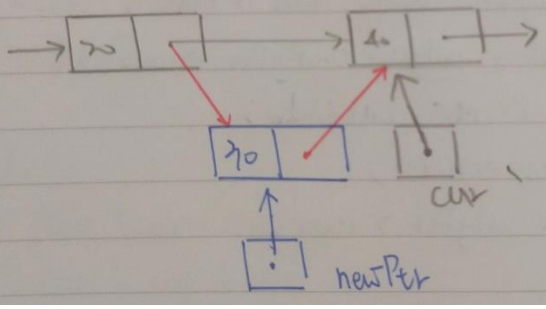
```
curr -> next = NULL
delete curr;
curr = NULL;
```

歸還空間給系統
順序不可換 要delete完才能改NULL

Insert 插入

```
newPtr -> next = curr;
prev -> next = newPtr;
```

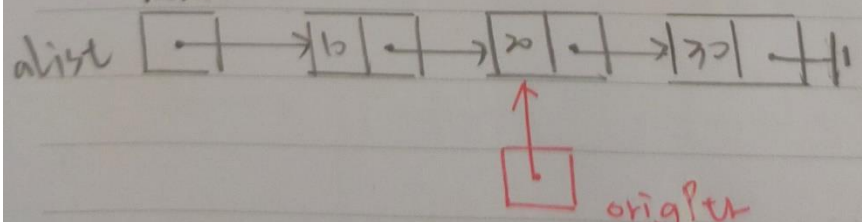
順序可換

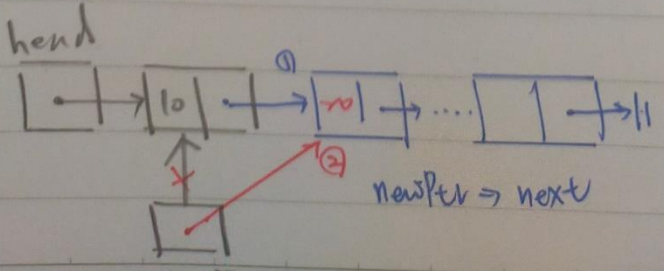


Subject : Pointer ADT List Date : ...../...

Deep copy.

```
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;
    } // for
    newPtr -> next = NULL;
```



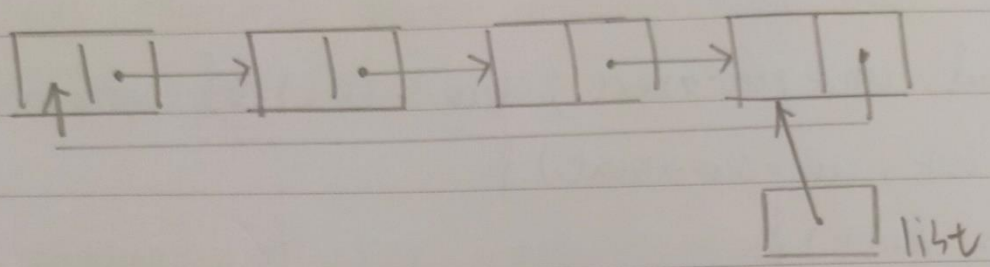


Subject : Circular Linked List

Date : ..

* Always to last node

make external pointer point to last node



Ex. Display :

```
if (list != NULL) {
    Node *first = list -> next;
    Node * cur = first;

    do {
      show (cur-> item);
      cur = cur -> next;
    } while (cur != first);
} // if
```

Subject : ✡ Dammy Head Node ✡ Date : ...../...../.....

✡ 讓 head 指向一個空的 head

✡ delete . insert 不用處理 special case

```
Node *prev, *cur;
for (prev=head, cur=prev->next; (cur != NULL) && (newValue> cur->ite
    ; prev = cur, cur = cur->next) {
  if (cur != NULL) {
    prev -> next = cur -> next;
    cur -> next = NULL;
    delete cur;
    cur = NULL;
  } // if
} // for
```

CH4

Subject : CH4 defining languages Date : ...../...../.....

The Basics of Grammers recognition algorithm 辨識演算法

- A C++ identifier begins with a letter and is followed by zero or more letters and digits.

- language of C++ identifiers
  Ids: { w = w is a legal C++ identifier }

- Grammer
  <identifier> : ① <letter> 字母 | ② <identifier><letter> | ③ <identifier><digit> 數字
  5X : A2C → ② <A2><C> → ③ <A><2> → ① <A>
  <letter> = a | b | ··· | z | A | ··· | Z |
  <digit> = 0 | 1 | ··· | 9 |

Palindromes

- language :
  Palindromes : { w = w reads the same left to right as right to left }

- Grammer
  <pal> = empty string | <ch> | a <pal> a | b <pal> b | ··· | z <pal> z
  <ch> = a | b | ··· | z | A | ··· | Z |

- recognition algorithm

```
isPal (in w = string) = boolean
  if (w is the empty string or w is of length) // base cas
     return true;
  else if (w's first and last characters are the same letter)
     return isPal (w minus its first and last characters)
  else return false;
```

$A^n B^n$

- The string that consists of n consecutive A's followed by n consecutive B's

- language : $L = \{w : w$ is of the form $A^n B^n$ for some $n \geq 0\}$
- Grammer : <legal-word> = empty string | A <legal word> B
- Recognition Algorithm:

```
isA^nB^n (in w : string) = boolean
  if (the length of w is zero) return true;
  else if (w begins with 'A' and ends with 'B')
     return isA^nB^n (w minus first and last);
  else return false;
```

Subject : 算式運算的表示法 Date : ...../...../.....

Algebraic Expressions

Infix expressions: An operator appears between its operands

ex: a+b

Prefix expressions: An operator appears before its operands

ex: +ab

Postfix expressions: An operator appears after its operands

ex: ab+

Fully Parenthesized Expressions

- Grammer:

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

<operator> = + | - | * | /

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

- convert a fully parenthesized infix expression to a postfix form

⇒ remove the parenthesis

ex: Infix expression ((a+b)*c) (+ab)*c) (*(+ab)c) *+abc

Prefix expression ab+c*

convert infix ⇒ postfix

⇒ move each other to the position marked by its corresponding closing parenthesis ⇒ remove the parenthesis

ex. (a+b)*c a.b.+.c.*

(ab+)*c

(ab+)c*

Double A

Subject :

Pretix

<pretix> = <identifier> | <operator> <pretix> <pretix>

<operator> = + | - | * | /

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

infix → pretix

Base case : one lowercase letter is a pretix exp.

recursive : <operator> < pretix> <pretix>

pretix → postfix

postfix (exp) = postfix (prefix) + postfix (prefix 2) + <operator>

infix → pretix &x. + * ab | cd

```
endPre (in first = integer) = integer
    last = strExp.length() -1 ; 最常前序式的節尾必須是該字串結尾
    if (first < 0) or (first > last) return -1;
    ch = strExp [first];
    if ( ch is an identifier) return first ;
    else if (ch is an operator) { + ⊛ 的前序
        firstEnd = endPre (first +1)
        if (firstEnd > -1) return endPre (firstEnd + 1);
        else return -1;
} // endPre
return -1;
```

Postfix

<postfix> = <identifier> | <postfix> <postfix> <operator>

Subject : Backtracking - Eight queen

No. :

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

- Backtracking : A strategy for guessing at a solution and backing up when impasse is reached

Solution :

- Base Case : If there are no more columns to consider
  ⇒ finished

- Recursive step : If you successfully place a queen in the current column ⇒ consider next column
  If you cannot place a queen in the current column ⇒ backtrack

Implementing 8 Queens

- object 1 : Board
  contains a vector of pointers to Queen objects on the board
- object 2 : Queen
  keeps track of its row and column placement

以遞迴搜尋2點間的路徑 HP Air

→ High Plaines Airline Company (HP Air)

→ The flight map for HP Air is a graph 記點和點的關係

Search R (in : originCity : City, in destinationCity : City)
: boolean

Mark originCity as visited
if (originCity is destinationCity)
  Terminate - the destination is reached
else
  for (each unvisited city C adjacent to originCity)
    searchR (C, destinationCity)

<Solution>:

```
maskVisited (originCity);
if (originCity == destinaCity) return true;
else {
  done = false;
  success = getNextCity (originCity, nextCity);
  while ( success && !done) {
    done = isPath (nextCity, destinationCity);
    if (!done) success = getNextCity (originCity, nextCity);
  } // while
  return done;
} // else
```

Subject : Pretix Expression 數學歸納法

No. :

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

• If E is a pretix expression

If Y is any nonempty string of nonblanks

Then E.Y cannot be pretix -但前序式後面再接上非空字串一定不是前序式

<prove>

Basis |E| = 1 : E is a single letter

<最簡> ⇒ EY does not begin with an operator

⇒ EY is not pretix

Inductive hypothesis : 1 < |E| < n

<假設> if E is pretix, then EY is not pretix

Inductive step : |E| = n : E = op $P_1 P_2$, where both P_1 P_2

「歸納」 are pretix expressions

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

反証 : Assume EY is pretix, then EY = op W_1 W_2, where

W_1 W_2 are pretix ⇒ $P_1 = W_1$

⇒ $P_2 Y = W_2$ 前序 ⇒ contradiction ⇒ EY is NOT pretix

加-但未必不是前序