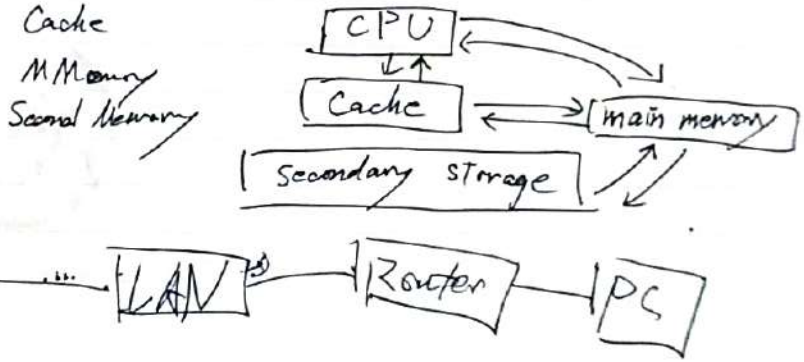
資料結構

Chapter 1 Recursion.

- Recursive Functions.
  - Factorials ! 階乘
  - Greatest Common ~~Denicter~~ Divisor
  - Search in array
  - Fibonacci Series
  - Combinatorial numbers 組合數
  - Towers of Hanoi 河內塔

⇒ Break problem into smaller identical problem.
(Divide AND Conquer)

CPU
Cache
MMemory
Second Memory



ex: print a string backward

→ print last character
→ remove last character
→ call function.

```
void WriteBackward( string s, int size ){
    if (size > 0 ) {                          // pos  // amount of char
        cout << s.substr( size-1, 1);
        writebackward ( s, size-1);
    }
}
```

Way 1

Way 2

```
Void writeArrayBackward ( const char anArray[], int first, int last){
    if ( first <= last) {
        cout << anArray[ last ] ;
        writeArrayBackward(anArray , first, last-1);
    }
}
```

ex. given a, b, add a to b and return a sum.

```
void sum ( int a, int b){
    if ( a ≤ b ) {
        return sum ( a+1, b) + a ;
    }else {
        return a
    }
} // sum
```

ex. GCD

Greatest Common Divisor

- gcd1 (x, y) = x if y = 0
  = gcd1 (x, y mod x) if y > x
  = gcd1 (y, x mod y) if else.
- gcd2 (x, y) = y if x mod y = 0
  = gcd (y, x mod y) otherwise.

x = 9 y = 6
x % y = 3
gcd(6, 3)
gcd(3, 6 % 3) ← new y = 0
↓ return of y

gcd2

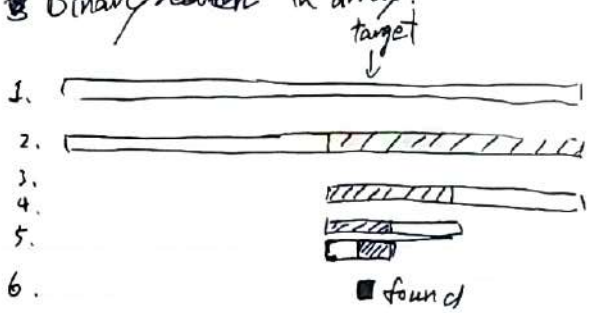
```
int gcd2 ( int x, int y) {
    if (x % y) {
        return y
    } else {
        return gcd2 ( y, x % y)
    }
} // end gcd2
```

gcd1

```
int gcd1 (int x, int y ) {
    if ( y == 0) {
        return x
    } else if (x < y){
        return gcd1 ( x, y mod x)
    } else {
        return gcd1 ( y, x mod y)
    }
}
```

x = 9 y = 6
⇒ gcd1 (9, 6)
⇒ gcd1 (6, 9 mod 6) 3
⇒ gcd1 (3, 6 mod 3) 0
⇒ y == 0, return 3

Binary Search in array:



Recursive Solution for finding max item in array.

```
if (anArray has only one item) {
    return maxArray(anArray) is the ~~only~~ item in anArray.
else ~~if~~ if(anArray > 1 item) {
    return max( maxArray(leftHalf of anArray),
                maxArray(rightHalf of anArray) )
}
```

Find k^{th} smallest item

- need pivot item
- cleverly arranging / partitioning the items in the array about this pivot.

• 只需找其中一邊 • better efficiency.

```
kSmall(k, anArray, first, last)
 = kSmall(k, anArray, first, pivot~~Point~~Index - 1)   左半邊
        if k < pivotIndex - first + 1
 = p    if k = pivotIndex - first + 1                   終止條件
 = ~~kSmall(k, anArray, pivotIndex-1)~~
 = kSmall(k - (pivotIndex - first + 1), anArray,        右半邊
        pivot~~Index~~ + 1, last)
        if k > pivotIndex - first + 1
```

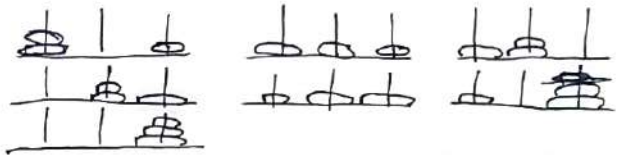


```
int kSmall(int k, string anArray, int first, int last) {
    if ~~~~
```

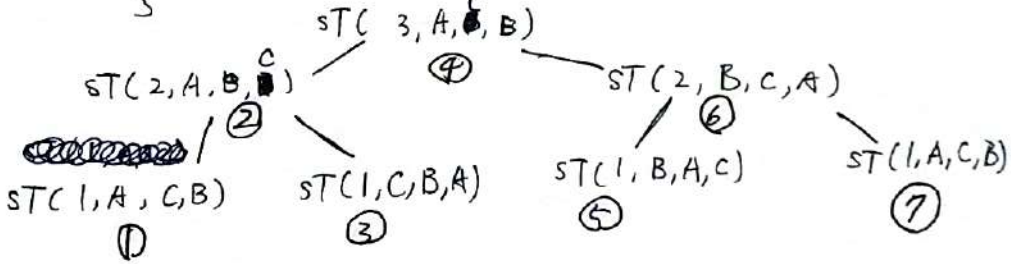
Tower of Hanoi 河內之塔.

for stack of 2, move ~~time~~ Time: $3 = 2^2 - 1$
for stack of 3, move time: $7 = 2^3 - 1$
etc.

Stack of 3:



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



Binary Recursion 二次遞迴

- 2 Recursive call for 1 non-base case.

刻度尺 - 劃 4個虛線前中間有個3連線。

- 劃 3 "-" 前劃 2 "-"
- 劃 2 "-" 前，畫 1 "-"
- base case : 1個 "-"

```
drawTicks(length){          ⇐ 畫 N 、刻度
    if (length > 0){
        drawTicks(length-1)
        draw ticks with length.
        drawTicks(length-1)
    }
}
```

```
void drawRuler (int nInches, int majorLength){
    drawOneTick(majorLength, 0);      ⇐主刻度
    for(int i = 1; i <= nInches; i++){
        drawTicks(majorLength-1);     主刻度
        drawOneTicks(majorLength, i);
    }
}
```

把一個數a加到b.

- linear Recursion.

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

- binary recursion. ↙ amount to add : n = b-a+1

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

linear called 8 times. , 7 times addition.
binary called 15 times. , 7 times addition.
which is more efficient ?? depend on the prespective.

Rabbits

month n.

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

rabbit(2) = 1
rabbit(1) = 1

Fibonacci sequence. 1, 1, 2, 3, 5, 8, 13, 21, 34...
n^{th} num equals to the sum of the 2 numbers bf n.

⇒ low efficiency.

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

Fibonacci Sequence.

if n = 5, how many times did rabbit call?

rabbit(5) → rabbit(4) → rabbit(3) → rabbit(2)
rabbit(2) rabbit(1)
rabbit(3) → rabbit(2)
rabbit(1)

called 9 times

n_k be num of recursive calls.

$n_1 = 1$ $n_2 = 1$
$n_3 = n_1 + n_2 + 1 = 3$
$n_4 = n_3 + n_2 + 1 = 5$
$n_5 = n_4 + n_3 + 1 = 9$
$n_6 = n_5 + n_4 + 1 = 15$
$n_7 = n_6 + n_5 + 1 = 25$
$n_8 = n_7 + n_6 + 1 = 41$

n_k ~~double~~ at least other time.
DOUBLE
隨 k 增加呼叫次數大增

☆ $n_k \geq 2^{\frac{k}{2}}$, exponential growth.
可用空間換時間，把算到的答案存下，下次可查表。

- Linear recursion Fibonacci

```
Algorithm linearFibonacci(k) {
  if (k=1)
    return (k,0)
  else
    (i,j) = linearFibonacci(k-1)
    return (i+j, i)
}
```

☆ 呼叫次數線性成長

Called 5

get (5,3) ← F(5) call→ F(4) call→ F(3) call→ F(2) call→ F(1)
return (3,2) return (2,1) return (1,1) return (1,0)

巨匠專業 有口皆碑 www.pcschool.com.tw

Computing Power. nth times.

$x^0 = 1$
$x^n = (x^{\frac{n}{2}})^2$ for even n.
$x^n = x \cdot (x^{\frac{n}{2}})^2$ for odd n.

```
double power ~~nTimesAlgorithm~~ ( double x, int n) {
  ~~if (n/2 %% 0) { // even~~
  ~~nTimesAlgorithm(x, n/2)~~
  if ( n==1)
    return X;
  else {
    double halfpower = power( x, n/2);
    if (n%2==0) { //even
      return (halfpower * halfpower)
    }else // odd
      return x * halfpower * halfpower )
  }
} // big else.
} // power
```

{ how many recurise calls?
{ how many multiplication?

ex: 9^4 ⇒ power(9,4)

power(9,2) ⇒ $81 \times 81 = 81^2 = 9^4$
2 multiplication
3 recursive calles

k multiplication count

9^{32} ⇒ 32(1) → 16(1) → 8(1) → 4(1) → 2(1) →
→ (1)(2) → 0(0)
17 times multiplication

9^{19} ⇒ 19(2) → 9(2) → 4(1) → 2(1) → (1)(2) → 0(0)
8 times multiplication

Ex. 哪個整數的平方最接近A小於30

getValue (int first, int last, int n)

getValue (1, 30, 30)

```
int getValue ( int a, int b, int n ) {
    int returnValue;
    int  c = (a+b)/2;
    if ( c*c<=n) && ( n<((c+1)*(c+1)))
        return value = c;
    else if (c*c>n){
        returnValue = getValue ( a, c-1, n); }
    else {
        returnValue = getValue ( c+1, b, n);
    }
~~// getValue~~
return returnValue;
} // getValue.
```

Ex Practice 15.

$$Acker(m,n) = \begin{cases} n+1 & \text{if } m=0 \\ Acker(m-1,1), & \text{if } n=0 \\ Acker(m-1, Acker(m,n-1)), & \text{otherwise} \end{cases}$$

what is (1,2)

A(0, A(1,1)) → A(0, 3) = 4

~~A(0,0)=1~~ ← A(0, A(1,0)) ⇒ A(0,2) ⇒ 3

A(0,1) ⇒ 2

Ex.

Mr. Spock's Dilemma

choosing k out of n things

$_k C_n$.

in terms of Earth:

C(n,k) = the number of groups of k planets that include Earth

↓

\+ the number of groups of k planets that do not include Earth.

C(n,k) = ways of choosing k-1 out of n-1 things

\+ ways of choosing k out of n-1 things

↓

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

what's the base case?

C(1,1) = 1 , C(k,k) = 1 only one choice.

C(n,0) = 0 don't go, so only one choice.

C(n,k) = 0 if k>n 怎麼.

---

ex: C(4,2)

C(3,1) + C(3,2)

C(2,0) + C(2,1) C(2,1) + C(2,2)

1 C(1,0) + C(1,1) C(1,0) + C(1,1) 1

1 1 1 1

result: 6 combination.

total recursive called: 11

- 2C(n,k)-1 = num of recursive calls because base case=1 and ... ☞ |leaf node| - |internal node| = 1

Observation from C(n,k)

- base case return 1
- num of recursive calls to non-base cases is equal to |internal node| = C(n,k) − 1.

Chapter 2.

Data Abstraction.

Classes → encapsulation, inheritance.

Cohesion - module performs a single task well. (tightly)
Coupling - measure of dependence among modules. (loosely)

Operation Contracts

1. Document limitation of the method.
2. Specify data flow
3. Do not specify how module will perform its tasks.
4. Specify pre- and post- condition.
5. unusual conditions
   - something that causes problems / throw exception.
6. begin the contract during analysis, finish during design.
7. used to document code, particularly in header file.

Key Issues in programming.

1. Modularity 模組化
2. Style
3. Modifiability
4. Ease of Use
5. Fail-safe programming
6. Debugging
7. Testing

ADT : modularity

- 模組化 . systematically control
- 可以把問題獨立出來，方便解決
- eliminate redundancies 重複
- 好讀、容、改

2 Solution : Cohesion or Coupling
高內聚 低耦合

ADT : Function abstraction 函式/功能抽象化

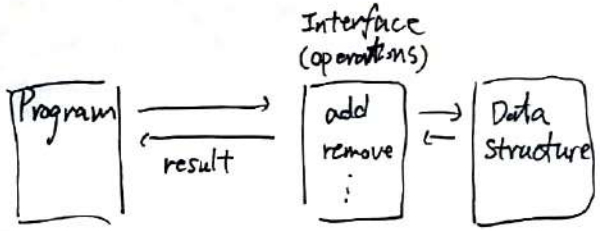
- 描述 & 實作分開
- module specification 講 module 要怎麼運作
- 資訊隱藏 & 實作隱藏
- 可更未來換成更佳的寫法

ADT: Goals.

- Do operation
  - Insert, Deletion, Search, Modify, etc. interface.
- Data Abstraction
  - allow each data structure in relative isolation from rest of the solution.
  - natural extension of functional abstraction.

ADT : 有 Data / Collection of Data.

- have operations
- 描述 . what it does
- 實作 , how it does it. & include specific data structure.



Implementation.

- implementation detail (data structure, how, etc.) show be hidden behind ADT Operations.

C++ Classes.

- Class define a new data type, etc.
- contain data members and methods.

header file : Classname.h
implementation : Classname.cpp. #include "classname.h"

Constructor → initializes the object with/without value.

- scope resolution operator "::"

e.g. , Sphere :: getRadius () { }
↳ a operation outside declaration of the class.

e.g an initilizer

```
Sphere :: Sphere() : theRadius (1.0)
{ }
```

Destructor

destroys an instance of an object when object's life time ends.

Inheritance

Super Class (base class) —inherit to→ Sub class (derived class)
public methods and data member

method defined / implement outside Class.

```
vOid Rational :: setRational (long n, long d) {
    if (d==0){
        error ("ERROR")
    }
    :
} // end setRational
```

Overloading & Overriding

```
Class something {
        func1 (int a) { ... }
        func1 (string b) { ... }   <- Overloading.
        :
}
```

=> same function

```
Class something2 : public something {
        func1 (long, long);   <= Overriding
        :
}
```

C++ namespace ☆

- 由 mechanism for logically grouping declarations and definitions into a common declarative region.

cx:
```
namespace myNamespace {
    // declarations
    int a;
}
```

cx:
```
using namespace std;
```
→ allow names of the elements to be used directly.

ex:
```
myNamespace :: a
```
→ Access element outside the namespace using scope resolution operator (::)

C++ standard library.

- item create in ↑ are declared in std namespace.
- I/O function from C++ library.
  => #include <iostream.h>

Exceptions

→ handle error in execution.

→ throw exception to indicate an error has occured.

→ try - catch ☆ throw ☆

```
try {... throw(type)...}
catch( exption class type1 ) {
      statements ...
}
catch ( type2 ){
      ----
} !
:
```

→ Exception Occured!!

1. 跳出 try , local variable ← destructor.
2. catch block run
3. go to last line of try block related catch block
4. continue to run rest of the code.

→ THROW exception class (" string argument")

- method that throws exception have a throw clause.

ex: void myMethod (int x) throw (Myexception) {
    if ( )
        throw Myexception(" ---- ")
}

- use std library exception or define yr own.

Define exception.

ex:

```
# include <stdexcept>
# include <string>
using namespace std;
```

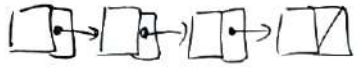
inherit from C++ std

①
```
class ListIndexOutOfRangeException: public out_of_range
{
    public :
        ListIndexOutOfRange (const string & message = "")
            : out_of_range (message.c_str())
        {}
}; // end.
```

②
```
class ListException : public logic_error {
    public :
        ListException ( const string & message = "")
                : logic_error (message.str()) {};
} // end
```

⇒
```
if ( size > MAX_LIST) {
    throw ListException ("ListException: " +
                         "List full on insert")
```

Unit 3. Linked List.



- do not require the shifting of items during insertions/deletion.
- dynamically ~~memory allocation~~. expand the list / a collection of items.

Pointer.

int * p;
- initially undefined, not null
- static allocation.

- pointer has location (address) in memory

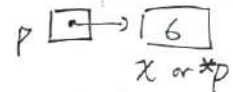
1. address-of operator.

new
```
int x; int* p;
p = &x;   把 x's 地址給 p.
```

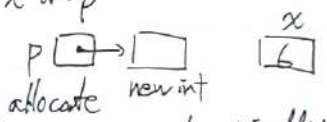
2. p = new int; 申請新空間,
   dynamically allocate a memory cell of size int.
   要經過 new 才能正常使用記憶體.
   note. can't allocate!! throw exception
   std:: bad_alloc(in the <new> header)

3. delete p;
   p = NULL; ← just to be save. 做個遺代.
   - "delete" can return dynamically allocated memory to the system for reuse. leave variable undefined.

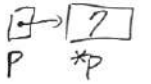
ex:
```
int* p, *q;
int x;
p = &x;   → get x's address.
*p = 6;   → put 6 in x's address
```



get memory → p = new int;



give value → *p = 7;

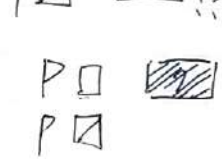


memory dynamically.

wrong delete → p = NULL;

memory leak!!!

correct → delete p;
p = NULL;



deallocating memory.

△ call p after delete?? dangling reference (illegal access)

```
delete p;
cout << *p << endl;  ← wrong!
```

動態陣列

Dynamic Allocation of Arrays.

```
int arraySize = 50;
double *anArray = new double[arraySize];
```
一個指標指向 50個空間

- array name is the pointer to its first element.
  anArray[2] = *(anArray+2) 有*才是取內容.

- 可主導的空間

```
double * old = anArray;
anArray = new double[3*arraySize];  ← 新空間
for( int i=0 ... ) {
    anArray[i] = old[i];   ← move old data to new space.
}

delete [] old;  ← deallocate a bunch of space.
```

saveFile


```
#include <cstdio>
#define SID_LEN 12
#define SIR_NUM 5
typedef struct student {
    char sid[SID_LEN];
    int score;
} studentType;

int main(void){
    FILE *outFile = NULL;
    ;
    outFile = fopen(fileName.c_str(), "a");   // write mode
    if(outFile != NULL)
        saveFile(outFile, allS, SR_NUM);
    return 0;
}
void saveFile(FILE *fp, studentType dA[], int no) {
    for(int i=0 ; i<no ; i++) {
        fwrite(&dA, sizeof(dA[i]), 1, fp);
        cout << dA[i].sid << "," << dA[i].score << endl;
    }
    fclose(fp);
}
```

算出需要的位元組

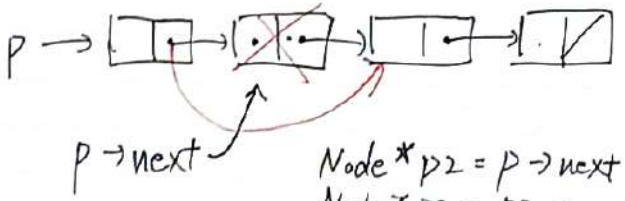
```
fseek(infile, 0, SEEK_END);
studentNo = ftell(infile) / sizeof(studentType);   ← 看幾筆
rewind(infile);
```

Pointer based Linked List.

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

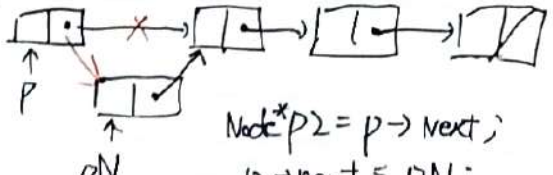
has header pointer
`Node *p;`
get member in node
`p → item.`

deletion.




```
Node *p2 = p → next → next;
Node *p3 = p2 → next;
delete p2;   p2=NULL;   avoid dangling reference.
p → next = p3;
```

insertion.




```
Node*p2 = p → next;
p → next = pN;
pN → next = p2;
```

visit a sorted linked list for insert/delete.
到底才能做事.

```
Node *prev, *cur;
for( prev = Null, cur = head; cur != NULL) && (newValue >
        cur → item); prev = cur ; cur = cur → next );
```

delete

```
if (prev = NULL) {
    head = (cur → next);
}
else prev → next = cur → next ;
cur → next = NULL;
delete cur;
cur = NULL;
}
```

insert

```
if (prev == NULL) {
    newPtr → next = head;
    head = newPtr;
}
else {
    newPtr → next = cur;
    prev → next = newPtr;
}
```

My Notes

Important Concepts worth keeping

Today: / /

Pointer - base List implementation

```
# include "ListExcept.h"
# include "ListIndexOutOfRangeException"

typedef int ListItemType
class List {
    public:

    List();
    bool  isEmpty() const;
    int   getLength const;
    void  insert( int index, const ListItemType & newItem)
          throw (ListIndexOutOfRangeException, ListException);
    void  remove(int Index) throw (ListIndexOutOfRangeException);
    void  retrieve(... )...;
    Struct Node{
       ListItemType  item;
       Node *next;
    }

    Node *find( int Index) const;  //找位置
}
```

dynamically allocated memory 需要自己寫 destructor !!

```
List :: ~List() {
      while (!isEmpty())
          remove();
}
```

My Questions

Problems & Difficulties needing exploration

My learning weather report

shallow copy head 複製到，但資料還是一份

deep copy 所有東西有兩份

ex; goal. List x(y);

y is a existing list, x initializes with data in y.

```
List:: List( const &aList): size( aList.size)
{
   if (aList.head == NULL)
      head = NULL;
   else
   {
      head = newListNode;
      head→item = aList.head→item;
      for( ListNode* origPtr= aList.head→next;
           origPtr != NULL; origPtr= origPtr→next)
      {  newPtr→next = new ListNode;
         newPtr = newPtr→next;
         newPtr→item = origPtr→item; // deep copy
      }
      一個要數記2個地址( newPtr, newPtr→next)
      newPtr→next = NULL;
   } // end else ------ remember !!
}
} // end list
```

My Opinions

Thoughts, inspirations, and suggestions

密碼 cipher key

My Notes

Important Concepts worth keeping

Today: / /

Pointer - base List implementation

```
#include "ListExcept.h"
#include "ListIndexOutOfRangeException"

typedef int ListItemType;
class List {
    public:

    List();
    bool IsEmpty() const;
    int  getLength const;
    void insert( int index, const ListItemType & newItem)
         throw (ListIndexOutOfRangeException, ListException);
    void remove(int Index) throw (ListIndexOutOfRangeException);
    void retrieve(... )...;
    struct Node{
        ListItemType item;
        Node *next;
    }

    Node *find( int Index) const;  //找位置
}
```

dynamically allocated memory 需要自己寫 destructor !!

```
List :: ~List() {
        while (!IsEmpty())
            remove();
}
```


We are to learn, not just to be taught; Work smarter, not just harder.

My Questions

Problems & Difficulties needing exploration

My learning weather report

shallow copy head 複製到，但資料還是一份

deep copy 所有東西 有兩份

ex; goal. List x(y);
y is a existing list, x initializes with data in y.

```
List:: List( const &aList) : size ( aList.size)
{
    if (aList.head == NULL)
        head = NULL;
    else
    {
        head = newListNode;
        head→item = aList.head→item;
        for ( ListNode* origPtr= aList.head→next;
              origPtr != NULL; origPtr= origPtr→next)
        {   newPtr→next = new ListNode;
            newPtr = newPtr→next;
            newPtr→item = origPtr→item; // deep copy
        }
        一個要數記2個地址 ( newPtr , newPtr→next)
        newPtr→next = NULL;
    } // end else          remember!!
    :
} // end list
```

My Opinions

Thoughts, inspirations, and suggestions

來為求學，非僅受教；不以力搏，要以智取。—《三呆斜說》

My Notes

Important Concepts worth keeping

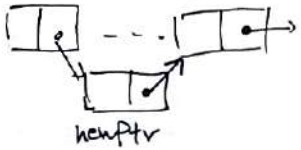
Today: ← find a specific node

```
List :: ListNode * find (int index) const
{
    if (( index < 1) || index > getLength() )
    {   return NULL;
    }
    else {
        ListNode * cur = head;
        for( int skip = 1 ; skip < index ; skip ++ )
            cur = cur -> next;
        retur cur; // the pointer
    } // else
} // end find
```

- insert 的部分，注意使用 exception for mem alloc, outofRange.

```
if( index == 1 ) {
    newPtr -> next = head;
} else {
    ListNode * prev = find (index - 1) ;
    newPtr -> next = prev -> next ;
    prev -> next = newPtr;
} // end else

catch( bad_alloc e) {
    ⋮
}
```



My Questions

Problems & Difficulties needing exploration

My learning weather report

Compare Array based and Pointer based.
AB PB

Size
- Increasing size of resizeable array waste time & storage. → requires copy
- Linked List grow & shrink as necessary

storage.
AB requires less memory than PB

Retrival
AB is faster → direct access using index i
PB is linear time depend on i

Insertion / Deletion
AB : need to shift data. action at head / front takes time.
PB : require traversal, constant time

My Opinions

Thoughts, inspirations, and suggestions

密碼 cipher key

My Notes

Important Concepts worth keeping

Today: / /

Linked list saved in a file.

- store only data, no pointers.
- treat first case differently.
- recreate list by placing each item at the end of the list.

Saving

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

Reading

```
ifstream inFile (fileName);
int nextItem;
if (inFile >> nextItem)
{
   try {  head = new Node;
          head->item = nextItem;
          head->next = NULL;
          tail = head;
          while ( inFile >> nextItem) {
             tail->next = new Node;
             tail = tail->next;
             tail->item = newItem;
             tail->next = NULL;
          } // while.
   } // try
   catch ( bad_alloc e) { }
} // end if
inFile.close();
```


My Questions

Problems & Difficulties needing exploration

My learning weather report

△ Passing a linked list to a Method.

- having access to head means having access to the whole list
- Pass head pointer to a method as a reference arguement
  → 才能改頭

△ ~~Test~~ Linked List ✗ recursion.

- sorted list??
  1. head == Null ;
  2. head->next == NULL ;
  3. head->item < head->next->item
     and head->next point to a sorted linked list
- Print list backward.
  - write the list minus its first item backward
  - write the first item backward.

△ Object as a linked list.

My Opinions

Thoughts, inspirations, and suggestions

```
typedef ClassName ItemType;
struct Node {
   ItemType item;
   Node*  next;
};
```

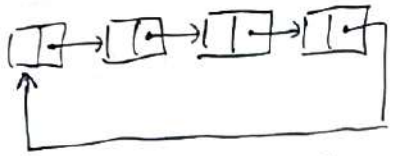
把多個同一個 class 的物件連起來.

密碼 cipher key

My Notes
Important Concepts worth keeping

Today: / /

△ Circular Linked List.



- no node in CList contains NULL
- make a Ptr to last node not head.

△ Dummy HeadNode.

- 去掉 head 的 special case.


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

delete {

```
if ( cur != NULL ) {
    prev->next = cur->next;
    cur->next = NULL;
    delete cur;
    cur = NULL;
}
```

}

```
} // for
```

insert {

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

}

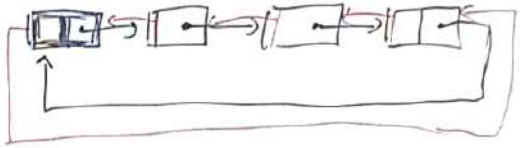
My Questions
Problems & Difficulties needing exploration

Doubly Linked List

My learning weather report

- each node point to both predeccessor & successor
- often has a dummy head node & circular to eliminate special case.

△ Circular Doubly linked list with dummy head node



delete

```
(cur->precede)->next = cur->next;
(cur->next)->precede = cur->precede;
```

Insert newPtr

```
newPtr->next = cur;
newPtr->precede = cur->precede;
(cur->precede)->next = newPtr;
(cur->precede) = newPtr;
```

My Opinions
Thoughts, inspirations, &