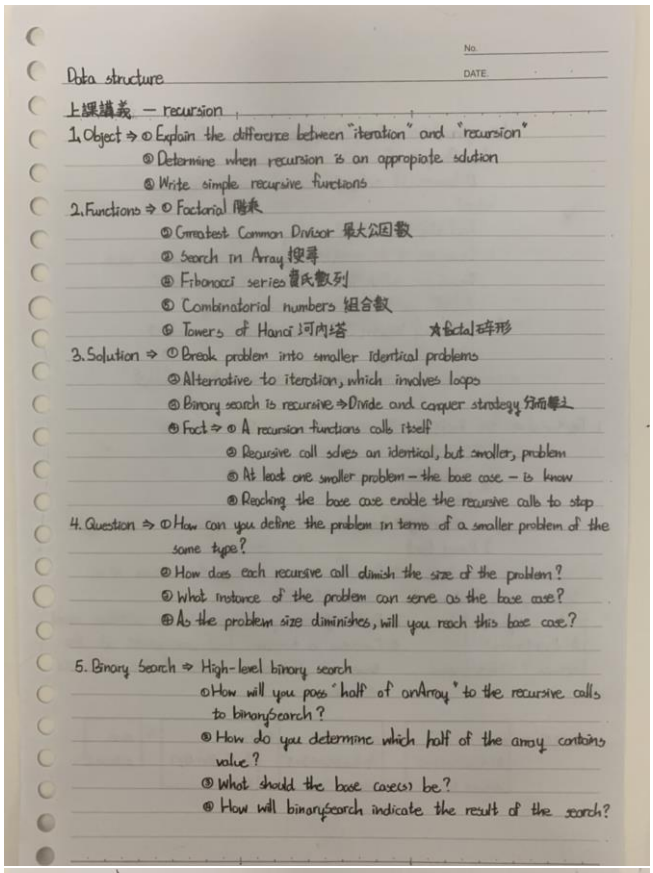
DS01 遞迴:



Data structure

上課講義 — recursion

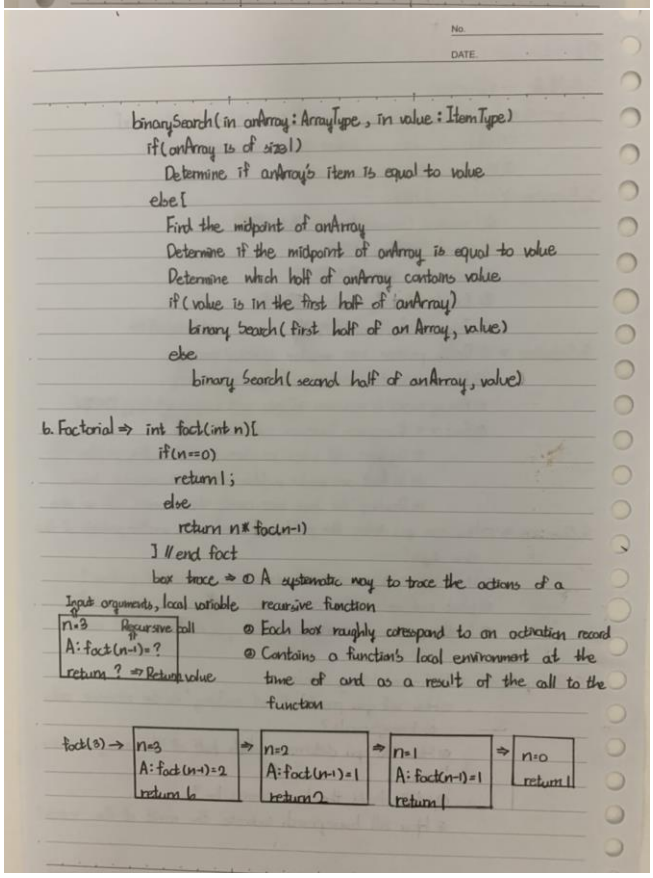
1. Object ⇒ ① Explain the difference between "iteration" and "recursion"
② Determine when recursion is an appropriate solution
③ Write simple recursive functions

2. Functions ⇒ ① Factorial 階乘
② Greatest Common Divisor 最大公因數
③ Search in Array 搜尋
④ Fibonacci series 費氏數列
⑤ Combinatorial numbers 組合數
⑥ Towers of Hanoi 河內塔 ★fractal 碎形

3. Solution ⇒ ① Break problem into smaller identical problems
② Alternative to iteration, which involves loops
③ Binary search is recursive ⇒ Divide and conquer strategy 分而治之
④ Fact ⇒ ① A recursion functions calls itself
② Recursive call solves an identical, but smaller, problem
③ At least one smaller problem – the base case – is know
④ Reaching the base case enable the recursive calls to stop

4. Question ⇒ ① How can you define the problem in terms of a smaller problem of the same type?
② How does each recursive call diminish the size of the problem?
③ What instance of the problem can serve as the base case?
④ As the problem size diminishes, will you reach this base case?

5. Binary Search ⇒ High-level binary search
① How will you pass "half of anArray" to the recursive calls to binarySearch?
② How do you determine which half of the array contains value?
③ What should the base case(s) be?
④ How will binarySearch indicate the result of the search?



```
binarySearch(in anArray: ArrayType, in value: ItemType)
  if(anArray is of size1)
    Determine if anArray's item is equal to value
  else{
    Find the midpoint of anArray
    Determine if the midpoint of anArray is equal to value
    Determine which half of anArray contains value
    if(value is in the first half of anArray)
      binary Search(first half of anArray, value)
    else
      binary Search(second half of anArray, value)
```

6. Factorial ⇒

```
int fact(int n){
  if(n==0)
    return 1;
  else
    return n*fact(n-1)
} // end fact
```

box trace ⇒ ① A systematic way to trace the actions of a recursive function
② Each box roughly correspond to an activation record
③ Contains a function's local environment at the time of and as a result of the call to the function

Input arguments, local variable

Review:

1. Fibonacci (費氏數列)

f(0) = 0 f(0) = 0
f(1) = 1 f(1) = 1
f(2) = 2 f(n) = f(n-1) + f(n-2) n>=2
f(3) = 3
f(4) = 5
f(5) = 8
f(6) = 13

Rabbit(兔子的故事)

r(1) = 1 1 (recursive calls)
r(2) = 1 1
r(3) = 2 2+1 = 3
r(4) = 3 3+1+1 = 5
r(5) = 5 5+3+1 = 9 ⇒呼叫次數double 變慢
r(n) = 1 if r<=2
= r(n-1) + r(n-2) + 1

---

linear recusion (線性遞迴) ⇒ better Fibonacci

Algorithm linearFibonacci(k)
Input: A nonnegative integer k
Output: Pair of Fibonacci numbers (F_k, F_{k-1})
if k=1, then
return(k, 0) // base cases: $k=1 \Rightarrow (F_1, F_0)$
else
(i, j) = linearFibonacci(k-1) // (F_{k-1}, F_{k-2})
return(i+j, i) //$(F_k = F_{k-1} + F_{k-2}, F_{k-1})$

---

2. TailRecursion

```
int iterativeRabbit(int n){
  int pre = 1  // n=1
  int cur = 1  // n=2
  int sum = 1
  for(int i=3; i<=n; i++){
    sum = pre+cur;
    pre = cur;
    cur = sum;
  }
  return sum;
}
```

pre cur sum
⇓ ⇓
pre cur

i=3 做3次fn法
i=4
i=5

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

n=5 ⇒ rabbit(4)①+rabbit(3)
②⇓ ⇓③
rabbit(3)+rabbit(2) rabbit(2)+rabbit(1)
④⇓ ⇓
rabbit(2)+rabbit(1)

Iterative V.S. Recursive

```
void writeBackward(string s, int size){
  while (size>0){
    cout << s.substr(size-1,1);
    size--;
  }
}
```

```
void writeBackward(string s, int size){
  if(size>0){
    cout << s.substr(size-1,1);
    writeBackward(s, size-1);
  }
}
```

Tail recursion ⇓
最後一次呼叫就是整個一組的合

Turn to TailRecursion

```
int fact(int n){
  if(n==0)
    return 1;
  else
    return n*fact(n-1)
}
```

⇒

```
int fact(int n, int result){
  if(n==0)
    return result;
  else
    return fact(n-1, n*result);
}
```

3. GCD (最大公因數)

遞迴

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

⇒

x=9, y=6, gcd1(6,3)=?3
⇓
x=6, y=3, gcd1(3,0)=?3
⇓
x=3, y=0, gcd1(3,0)=?3
return 3

⇒

x=9, y=6, gcd2(6,3)=?3
x=6, y=3 return 3

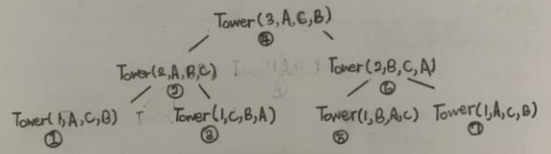
5. Binary Search with an Array (二元搜尋陣列)

```
int binarySearch (const int anArray[], int first, int last, int value){
  int index;
  if(first > last)
    index = -1;
  else{
    int mid=(first+last)/2;
    if(value == anArray[mid])
      index = mid;
    else if (value< anArray[mid])
      index = binarySearch(anArray, first, mid-1, value);
    else
      index = binarySearch(anArray, mid+1, last, value);
  }
  return index;
}
```

6.

4. Towers of Hanoi (河內塔)

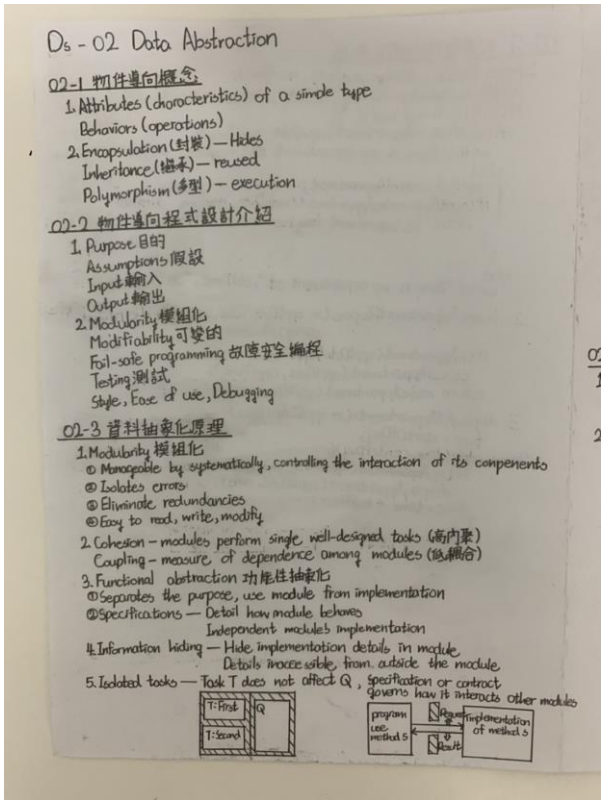
```
void tower(int n, char sou, char des, char aux){
  if(n==1)
    cout << "Move top disk" << sou << "to" << des << endl;
  else{
    tower(n-1, sou, aux, des);
    tower( 1 , sou, des , aux);
    tower(n-1, aux, des , sou);
  }
```



6. Finding k^{th} Smallest Item in an Array

```
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), anArry, pivotIndex+1, last)
```

DS02 資料抽象化:



Ds - 02 Data Abstraction

02-1 物件導向概念:

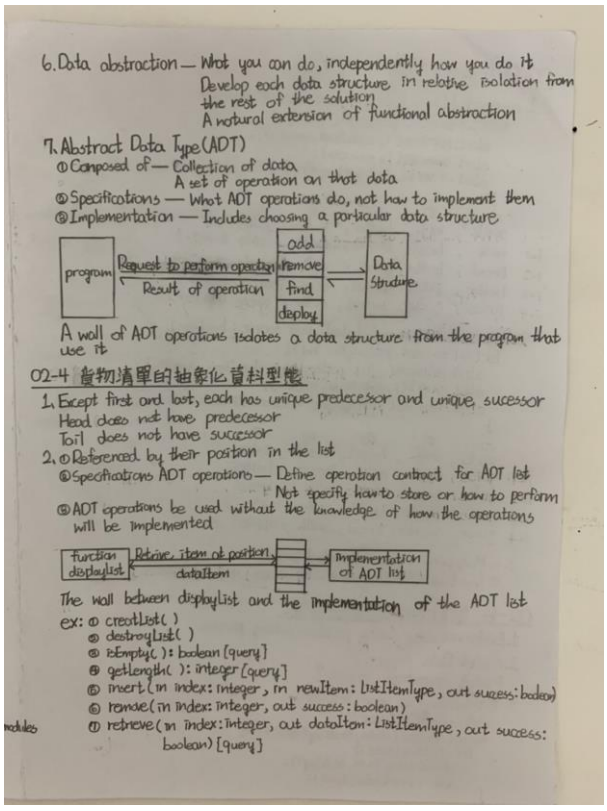
1. Attributes (characteristics) of a simple type
   Behaviors (operations)
2. Encapsulation (封裝) — Hides
   Inheritance (繼承) — reused
   Polymorphism (多型) — execution

02-2 物件導向程式設計介紹

1. Purpose 目的
   Assumptions 假設
   Input 輸入
   Output 輸出
2. Modularity 模組化
   Modifiability 可變的
   Fail-safe programming 故障安全編程
   Testing 測試
   Style, Ease of use, Debugging

02-3 資料抽象化原理

1. Modularity 模組化
   ① Manageable by systematically, controlling the interaction of its components
   ② Isolates errors
   ③ Eliminate redundancies
   ④ Easy to read, write, modify
2. Cohesion — modules perform single, well-designed tasks (高內聚)
   Coupling — measure of dependence among modules (低耦合)
3. Functional abstraction 功能性抽象化
   ① Separates the purpose, use module from implementation
   ② Specifications — Detail how module behaves
   Independent modules implementation
4. Information hiding — Hide implementation details in module
   Details inaccessible, from outside the module
5. Isolated tasks — Task T does not affect Q, specification or contract governs how it interacts other modules



6. Data abstraction — What you can do, independently how you do it
   Develop each data structure in relative isolation from the rest of the solution
   A natural extension of functional abstraction

7. Abstract Data Type (ADT)
   ① Composed of — Collection of data
   A set of operation on that data
   ② Specifications — What ADT operations do, not how to implement them
   ③ Implementation — Includes choosing a particular data structure

A wall of ADT operations isolates a data structure from the program that use it

02-4 貨物清單的抽象化資料型態

1. Except first and last, each has unique predecessor and unique sucessor
   Head does not have predecessor
   Tail does not have successor
2. ① Referenced by their position in the list
   ② Specifications ADT operations — Define operation contract for ADT list
   Not specify how to store or how to perform
   ③ ADT operations be used without the knowledge of how the operations will be implemented

The wall between displaylist and the implementation of the ADT list

ex: ① creatList( )
② destroyList( )
③ isEmpty( ): boolean [query]
④ getLength( ): integer [query]
⑤ insert (in index: integer, in newItem: ListItemType, out sucess: boolean)
⑥ remove (in index: integer, out success: boolean)
⑦ retrieve (in index: integer, out dataItem: ListItemType, out success: boolean) [query]

02-7 行事歷的抽象化形態

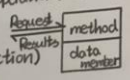
```
1. read(oldDate, oldTime, newDate, newTime);
   apptBook.checkAppointment(oldDate, oldTime, purpose);
   if(purpose is not null){
     if(apptBook.isAppointment(newDate, newTime))
       write("There is an appointment at", newTime, "on", newDate)
     else{
       apptBook.cancelAppointment;
       if(apptBook.makeAppointment(newDate, newTime, purpose)
         write("The appointment has been moved to", newTime, "on", newDate);
       }
     }
   else
     write("There is no appointment at", oldTime, "on", oldDate);

2. changeAppointmentPurpose(in apptDate:Date, in apptDate:Time, in purpose:
                            string):boolean
     if(isAppointment(apptDate, apptTime))
       cancelAppointment(apptDate, apptTime);
     return makeAppointment(apptDate, apptTime, purpose);

3. displayAllAppointments(in apptDate:Date)
     time = startOfDay;
     while(time < endOfDay){
       if(isAppointment(apptDate, time))
         displayAppointment(apptDate, time);
       time = time + halfHour;
     }
```

02-8 球體的C++類別

1. Choose the data structure to represent the ADT's data is a part of implementation
   Implementation detail should behind the wall of ADT operation
2. Encapsulation(封裝) — Combine an ADT's data with operations of object
   ① Object is an instance of class
   ② Class define a new data type
   ③ Contain data member and methods (member function)
   ④ All member are private, but can specify public
   ⑤ Encapsulation hides implementation details



3. Class definition in head file — Classname.h
   Class's method placed in — Classname.cpp
4. ex: const double PI = 3.14159265

```
class Sphere{
public:
  sphere( )                                  // Default constructor
  sphere(double initialRadius);              // Constructor
  void setRadius(double newRadius);
  double getRadius( ) const;                 //半徑
  double getDiameter( ) const;               //直徑
  double getCircumference( ) const;          //圓周
  double getArea( ) const;                   //面積
  double volume( ) const;                    //體積
  void displayStatistics( ) const;           //顯示資訊
private:
  double theRadius;          // Data member should be private 半徑
};
```

5. Constructors(建構)
   ① Creat and initialize new intance of the class
   ② same name at the class
   ③ Have no return type, not even void
   ④ A class have several constructors
   ⑤ Default constructor has no arguments
   ⑥ The compiler will generate a default constructor if do not define any constructors
   ⑦ The implementation of a method qualifies its name with the — scope resolution operator :: (範圍解析運算符號)
   ⑧ Sets data members to initial values, Can use an initializer
   ⑨ Sphere::Sphere( ):theRadius(1.0){
     }

6. Destructer (解構元)
① Destroys an instance of an object when the object's lifetime ends
② Can omit the destructor
③ Each class has one distructor
④ Compiler will generate a distructor if do not define one

7. @file Sphere.cpp

```
#include <iostream>
#include "Sphere.h"
using namespace std;
Sphere::Sphere(): theRadius(1.0){
}
Sphere::Sphere(double initialRadius){
  if(initialRadius>0)
    theRadius = initialRadius;
  else
    theRadius = 1.0;
}
void Sphere::setRadius(double newRadius){
  if (newRadius>0)
    theRadius = newRadius;
  else
    theRadius = 1.0;
}
double Sphere::getRadius() const{
  return theRadius;
}
double Sphere::getArea() const{
  return 4.0 * PI * theRadius * theRadius;
}
```

02-9 C++類別的繼承

1. Drived class or subclass inherit any of the publicly of base class or superclass

```
#include "Sphere.h"
enum Color [RED, BLUE, GREEN, YELLOW];
class ColoredSphere : public Sphere{
public:
  Color getColor() const;
private:
  Color c;
}
```

2. ① Derived class is considered to also be instance of the base class
② An instance of a derived class can invoke public methods of the base class

02-12 C++類別的多載

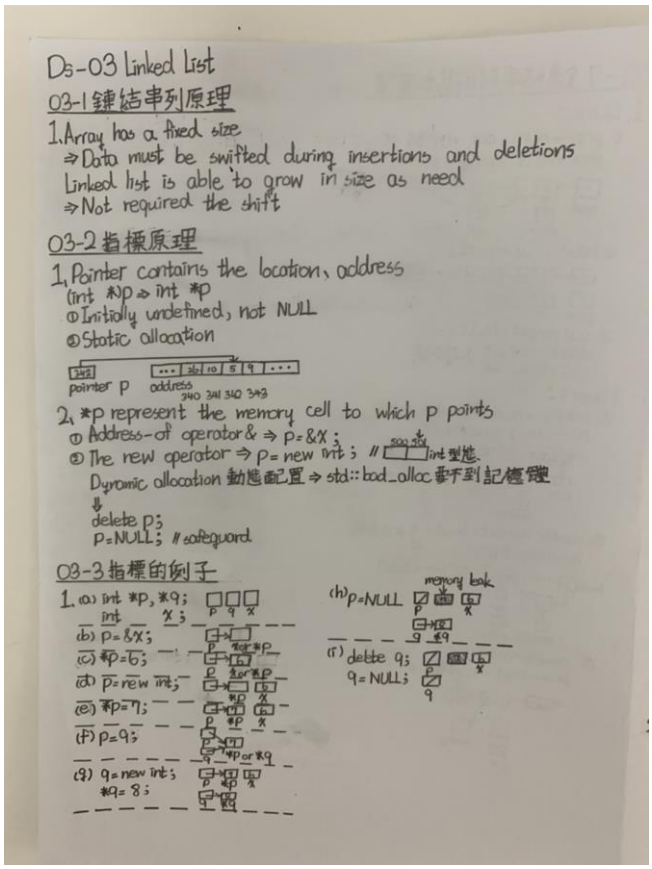
1.
```
class Rational{
public:
  Rational add(Rational);
  Rational add(long);
}
Rational Rational::add(Rational r){
  ...
}
Rational Rational::add(long i){
  Rational r;
  r.setRational(i,1);
  return add(r);
}
```

2.
```
class Rational{                    // Inheritance
  protected:
    long numerator;
    long denominator;
    void reduce(void);
  public:
    ...
};
class Integer: public Rational{
  public:
    void setRational(long, long)
    void setRational(long)
};
```

Private: only class instance
Protected: subclass instance
Public: any class instance

3.
```
void Integer::setRational(long num, long denom){
  if(num%denom != 0)
    error("ERROR: non-integer assigned to Integer variable");
  numerator = num/denom;
  denominator = 1;
}
void Integer::setRational(long num){
  numerator=num;
  denominator=1;
}
```

DS03 鏈結串列:



Ds-03 Linked List

03-1 鏈結串列原理

1. Array has a fixed size
   ⇒ Data must be swifted during insertions and deletions
   Linked list is able to grow in size as need
   ⇒ Not required the shift

03-2 指標原理

1. Pointer contains the location, address
   (int *)p ⇒ int *p
   ① Initially undefined, not NULL
   ② Static allocation

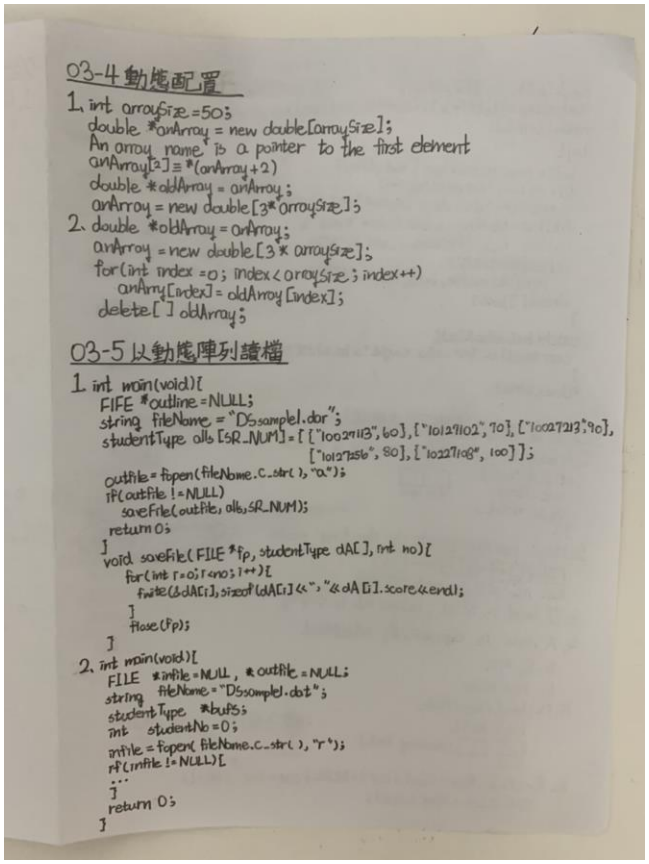
pointer p address 340 341 340 343

2. *p represent the memory cell to which p points
   ① Address-of operator & ⇒ p=&x;
   ② The new operator ⇒ p=new int; // int型態.
   Dyromic allocation 動態配置 ⇒ std::bad_alloc 要不到記憶體
   ⇓
   delete p;
   p=NULL; // safeguard

03-3 指標的例子

1. (a) int *p, *q; int x;
   (b) p=&x;
   (c) *p=6;
   (d) p=new int;
   (e) *p=7;
   (f) p=q;
   (g) q=new int; *q=8;
   (h) p=NULL
   (i) delete q; q=NULL;

memory leak



03-4 動態配置

1. int arraySize=50;
   double *anArray = new double[arraySize];
   An array name is a pointer to the first element
   anArray[2] ≡ *(anArray+2)
   double *oldArray = anArray;
   anArray = new double[3*arraySize];
2. double *oldArray = anArray;
   anArray = new double[3*arraySize];
   for(int index=0; index<arraySize; index++)
   anArry[index] = oldArray[index];
   delete[] oldArray;

03-5 以動態陣列讀檔

```
1. int main(void){
     FILE *outline=NULL;
     string fileName = "Dssample1.dat";
     studentType all[SR_NUM] = { ["10027113",60], ["10127102",70], ["10027213",90],
                                 ["10127256",80], ["10227108",100] };
     outfile = fopen(fileName.c_str(), "a");
     if(outfile != NULL)
       saveFile(outfile, all, SR_NUM);
     return 0;
   }
   void saveFile(FILE *fp, studentType dA[], int no){
     for(int i=0; i<no; i++){
       fwrite(&dA[i], sizeof(dA[i]), "<<", "<<dA[i].score<<endl;
     }
     fclose(fp);
   }
2. int main(void){
     FILE *infile=NULL, *outfile=NULL;
     string fileName="Dssample1.dat";
     studentType *buf5;
     int studentNo=0;
     infile = fopen(fileName.c_str(), "r");
     if(infile != NULL){
     ...
     }
     return 0;
   }
```

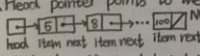
```
fseek(infile,0,SEEK_END);
studentNo = ftell(infile) / sizeof(studentType);
rewind(infile);
try{
 buffs = new studentType[studentNo];
 for(int i=0; i<studentNo; i++)
   fread(&buffs[i], sizeof(studentType), 1, infile);
 fileName = fileName.substr(0,8)+"2.dat";
 outfile = fopen(fileName.c_str(), "a");
 if(outfile!NULL)
   savefile(outfile, buffs, studentNo);
 delete[] buffs;
}
catch(bad_alloc &ba){
 cerr<<endl<<"bad_alloc caught:"<<ba.what()<<endl;
}
fclose(infile);
```

03-6 實做鏈結串列(以指標)

1. A node in a linked list usually a struct

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

2. Head pointer points to the first node



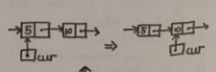
3. If head is NULL, linked list is empty
4. A node is dynamically allocated

```
Node *p;
p = new Node
```

5. (X) head = new Node

```
head = NULL
```

(memory leak)

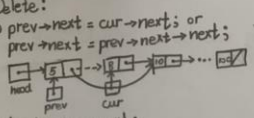


6. `for(Node *cur=head; cur!=NULL; cur=cur->next)`
   `cout<<cur->item<<endl;`

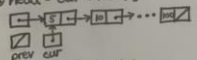
03-7 鏈結串列的基本運算

1. Delete:

① `prev->next = cur->next;` or `prev->next = prev->next->next;`



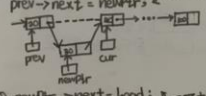
② `head = cur->next;`



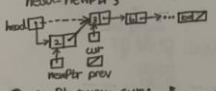
③ `cur->next = NULL;`
`delete cur;` 不可交換
`cur = NULL;`

2. insert:

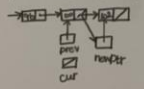
① `newPtr->next = cur;` 可交換
`prev->next = newPtr;`



② `newPtr->next = head;` 不可交換
`head = newPtr;`



③ `newPtr->new = cur;` 可交換
`prev->next = newPtr;`



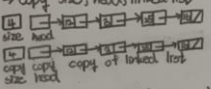
03-8已排序的鏈結串列

1. Node *prev, *cur
   for(prev=NULL,cur=head; (cur!=NULL)&&(newValue>cur->item);
       prev=cur; cur=cur->next);
2. C++ new and delete, dynamically and recycled
   Each pointer in a linked list is a pointer to the next node in the list
   Insertions and deletions involve traversing the list and pointer change
   special: ① Insert a node at beginning
            ② Delete the first node

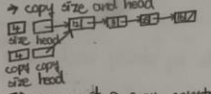
03-9 以指標實作鏈結串列

1.
```
#include "listException.h"
#include "listIndexOutOfRangeException.h"
typedef int ListItemType;
class List{
 public:
  List();
  List(const List& a List)
  ~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(int index, ListItemType &dataItem)const
    throw(ListIndexOutOfRangeException);
 private:
  struct ListNode{
   ListItemType item;
   ListNode *Next;
  };
  int size;
  ListNode *head;
  ListNode *find(int index)const;
 };
```

2. Default constructor initializes size and head
   A destructor is required for dynamically allocated memory
```
List::~List( ){
  while(!IS empty( ))
    remove( );
}
```
3. Copy constructor creats a (deep copy)
   ⇒ copy size, head, linked list



   In contrast, a (shallow copy)
   ⇒ copy size and head



   If you omit a copy constructor, the compiler generates one
   ⇒ only allocated arrays (只會在 a allow copy 一樣)

03-10 深層複製建構子

1.
```
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;
  }
```

03-12 比較不同方式的實作

1. Array-based implementation requies less memory
2. The time to access ith item
   ① Array : Const
   ② Pointer : Depends on I
   Insert and deletion
   ① Array : Shifting
   ② Pointer : traversal
3. Array : implicit , direct access an element
   pointer : explicit , traversal

03-13 鏈結串列的存檔

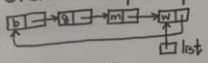
1. Use external file to preserve
   Write only data , no pointers
   Recreate the list from the file by placing each item at the end of the linked list
   ① Use tail pointer to adding nodes at the end
   ② First insertion as special case by setting the tail to head
2. ofstream outFile(fileName);
```
for( Node *cur = head; cur != NULL; cur = cur->next)
  outFile << cur->item << endl;
outFile. close( );
fistream 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->next = NULL;
    }
  catch (bad_alloc e){
  }
  inFile.close( );
```

03-15 鏈結串列的使用技巧

1. Data in a node of linked list can be an instance of a class
```
typedef ClassName ItemType;
struct Node{
  ItemType item;
  Node *next;
};
Node *head;
```

03-16 鏈結串列的各種變形

1. Circular Node
   ① last node points to the first node
   ② Every node has a successor
   ③ No node in a circular linked list contains NULL
   ④ Make external pointer point to last node instead of first node




```
if(lst != NULL){
  Node *first = list->next;
  Node *cur = first;
  do{
    show(cur->item);
    cur = cur->next;
  }while (cur != first );
}
```

2. Dummy Head Node
   ① Insert :
```
Node *prev, *cur;
for(prev = head, cur = prev->next; (cur != NULL) && (newValue > cur->item);
    prev = cur, cur = cur->next);
if(cur != NULL){
  prev->next = cur->next;
  cur->next = NULL;
  delete cur;
  cur = NULL;
}
```
   ②
```
for(prev = head, cur = pre->next;
    (cur != NULL) && (newValue > cur->item)
    ; prev = cur, cur = cur->next);
newPtr->next = cur;
prev->next = newPtr;
```