資訊一甲 **10927159** 林玟君

Data Structure (DS)

Lesson 1 : Recursion

- Practice 1-1 : Given two natural numbers a and b, where a > b, write a recursive function to compute the sum of all the integers from a to b, inclusively.

(sol)

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

- Four question / steps
  1. Define the problem in teams of smaller problems. // 定義
  2. See if a recursive call decreases the problem size // 簡化
  3. Find a complete set of base cases // 終止條件
  4. Every time it will always reach a base case // 保證終止

- Practice 1-2 : Greatest Common Devisor // 最大公因數

```
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 == 0) return y;
    else return gcd2 (y, x%y);
} // gcd2()
```

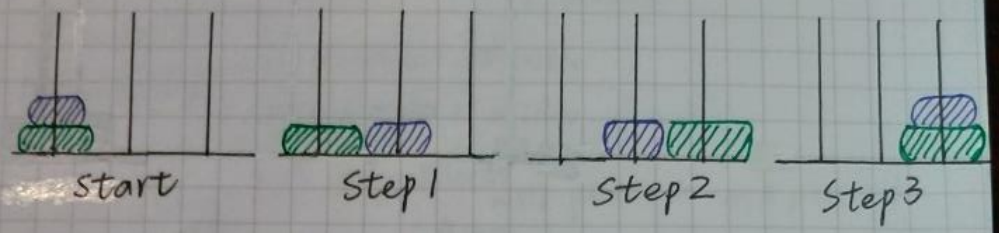
總結：2種方法皆可達到相同的效果
當 x>y，gcd2 比 gcd1 少做一次
當 x<y，2者次數一樣 // gcd2 要先讓 x>y
所以總體而言，gcd2 的效率更好(次數少)

- Binary Search with an Array

```
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);
    } // else
    return index;
} // binarySearch()
```

- Linear Recursion : Perform a single recursive call
  欠撣 - 遞迴

- Towers of Hanoi 河內塔




```
void solveTowers(int count, char source, char destination,
                                 個數        起點          終點
                 char spare ){
                      輔助
    if ( count == 1 )
        cout << "Move top disk from pole" << source
             << " to pole " << destination << endl;
    else {
        solveTowers (count-1, source, spare, destination); // X
        solveTowers (1 , source, destination, spare); // Y
        solveTowers (count-1, spare, destination, source); // Z
    } // else
} // solveTowers()
```

- Binary Recursion : Occurs whenever there are two recursive calls for each non-base case

- Practice 1-3. : Three ways to compute x^n for nonnegative integer n :

(a) Write an iterative function power1.

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

(b) Write a recursive function power2.

$x^0 = 1$; $x^n = x * x^{n-1}$, if $n > 0$.

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

(c) Write another recursive function power3.

$x^0 = 1$

$x^n = (x^{\frac{n}{2}})^2$, if $n>0$ and n is even.

$x^n = x * (x^{\frac{n}{2}})^2$, 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
} // power3 ()
```

(d) How many multiplication will each of the functions perform when computing 9^{32}; 9^{19};

	9^{32}	9^{19}
power1	32	19
power2	32	19
power3	7	8

(e) How many recursive call will power2 and power3 make when computing 9^{32}; 9^{19};

	9^{32}	9^{19}
power2	32	19
power3	7	6

recursion 寫的好，效能差很多.

- Practice 1-4 : What output does the following program produce?

```
int getValue ( int a, int b, int n) {
    int returnValue;
    cout << "Enter : a = " << a << " b=" << b << endl;
    int c = (a+b)/2;
    if ((c*c <= n) && (n<((c+1)*(c+1))))
        returnValue = c;
    else if ( c*c >n )
        returnValue = getValue (a, c-1, n);
    else returnValue = getValue ( c+1, b, n);
    cout << "Leave : a=" << a << "b = " << b << endl;
    return returnValue;
} // getValue()
int main() {
    cout << getValue (1, 30, 30) << endl;
    return 0;
} // main()
```

- Recursion and Efficiency : Tail recursion. 尾遞迴

效能好：遞迴 → 尾遞迴 → 迴圈

例：

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

↓ transfrom fact() into tail recursion (1)

```
int fact2(int n, int result){
    if(n==0) return result;
    else return fact2(n-1, n* result);
}// fact2()
```

Lesson 2 : Data Abstraction 抽象化.

□ Principles of Object-Oriented Programing

- Object-oriented language enable us to build class of objects (called instances)
- A class combines
  - Attributes (characteristics) of objects of a single type
    - Typically data
    - Called data member

    屬性
  - Behaviors (operations)
    - Typically operate on the data.
    - Called methods or member functions.

    運算
- 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 times.

- Modularity 模組化
  - Cohesion - modules perfrom single well-defined tasks
  - Coupling - measure of dependence among modules.

  課：高內聚 - 1個function處理少數的事，模組化
  低耦合 - 和其它函式傳遞少量參數，盡量獨立.

- Functional abstraction 功能性的抽象化
  - A module's specifications should 描述
    - Detail how the module behaves
    - Be independent of the module's implementation 實作

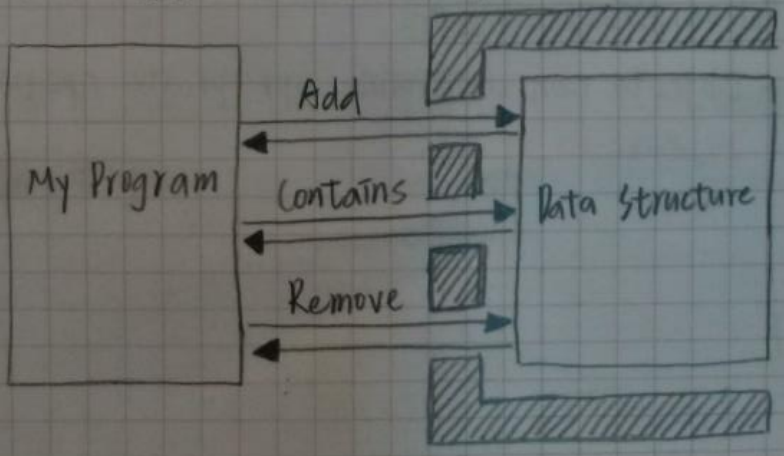
- Information hiding 資訊隱藏

□ Abstract Data Type : concepts

- The isolation of modules is not total.

□ Abstract Data Type : goals

- Typical operations on data.
- Data abstraction 資料抽象化



□ C++ Classes = Constructors

- Create and initialize new instance of a class.
- Have the same name as the class.
- Have no return type, not even void.(無宣告回傳型態)

■ A class can have several constructors

- A default constructor has no arguments 預設建構式
- The compiler will generate a default constructor if you do not define any constructions.

補:

Nullary 建構式 (無參數) 〈 自行撰寫無參數的建構式
編譯器自動加入的預設建構式 (default constructor)

知記憶體配置有關!

□ C++ Classes = Destructors 解構式

- Destroys an instance of an object when the object's lifetime end.

■ Each class has one destructor 預設解構.

- For many classes, you can omit the destructor
- The compiler will generate a destructor if you do not define one.

□ Inheritance in C++ 類別的繼承

■ An instance of a derived class can invoke public methods of the base class. 父類別 子類別

□ Overloading in Class Rational 類別的多載

- function的名稱一樣，傳入的參數不一樣

Private = only class instances
Protected = subclass instances
Public = any class instances

Overriding覆寫：子類別將父類函式重新定義以符合自身所需，達到多型的效果。

Overloading多載：簡化函式命名，不同式子共用相同的函式名稱。

- 降低所需命名的函式名稱
- 提高user的易用性

□ C++ Namespaces

- A mechanism for logically grouping declarations and definitions into a common declarative region.
- The contents of the namespace can be accessed by code inside or outside the namespace 範圍解析
  - Use the scope resolution operator (::) to access element from outside the namespace
  - Alternatively, the using declaration allows the names of the element to be used directly.
- Items declared in the C++ Standard Library are declared in the std namespace. 標準程式庫

□ C++ Exceptions 例外處理.

- A function can indicate that an error has occurred by throwing an exception.
- Uses a try block and catch blocks
  - try block = Place a statement that might throw an exception within a try block.

```
try {
    statement(s);
}
```

  - catch block = Deals with an eception

```
catch ( Exception class, Identifier) {
        statement(s);
} // catch()
```

Lesson 3 : Linker List. 鏈接串列

Array has a fixed size 陣列
- Data must be shift during insertions and deletions. (需要移動資料)

Linked list is able to grow in size as needed 鏈結串連(pointer)
- Does not require the shifting of items during insertions and deletions. (不需要移動資料)

□ Pointers

- Declaration of an interger pointer variable P : `int *P`
  - Initially undefined, but not NULL
  - Static allocation 一般變數：直接配給。靜態配置.
- To place the address of a variable into a pointer variable,
  - The address-of operator & : `P = &X`
  - The new operator : `P = new int;`
    - Dynamic allocation of a memory cell that can contain an interger. 動態配置
    - If the operator new can't allocate memory, it throws the expression std::bad_alloc (in the <new> header) 記憶體空間不夠！
- Delete 刪除. : `delete P;`

  → `P = NULL;` // safeguard
  // 避免之後再設同名pointer,
  // 會調用到錯誤的位址.
  // 甚至是別支程式的資料
  // 保持好習慣！

設為NULL之前，要先delete掉原本P占用的位址空間.
不然會造成記憶體浪費 (Memory leak)

□ Dynamic Allocation of Arrays. 動態(配置)陣列

Ex: int arraySize = 50;
double *anArray = new double[arraySize];

▲ 有new一個新東西，一樣要記得把原本占用的位址刪掉

Ex: double *oldArray = anArray;
anArray = new double[3*arraySize];
⋮
delete [] oldArray;

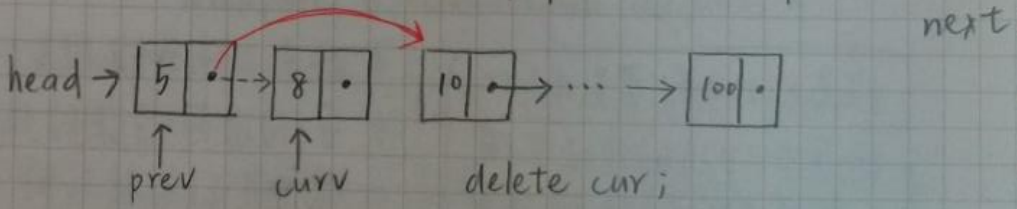
□ Pointer-Based Linked Lists

- A node in a linked list is usually a struct.

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

- The head pointer points to the first node in a linked list.
- If head is NULL, the linked list is empty.
- A node is a dynamically allocated.
- 鏈結串鏈的好處是散落在記憶體的不同位置，可不連號，配置陣列時必須是連號的!
- Deleting an interior node

prev->next = cur->next / prev->next = prev->next->next



delete cur;
cur = NULL;

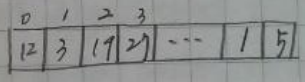
- Finding the point of insertion or deletion for a sorted linked list of objects.

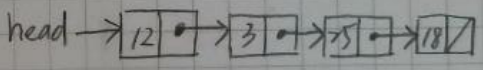
```
Node * prev, * cur;
for( prev = NULL, cur = head;
     (cur != NULL) && (newValue > cur→item);
     prev = cur, cur = cur→next );   // 走訪
```

□ Comparing Array-Based and Pointer-Based Implementations

Size
- Increasing the size of a resizable array can waste storage and time.
- Linked list grows and shinks as necessary.

Storage requirements Array-based implementation requires less memory than a pointer-based one for each item in the ADT.





指標每格都需拿多存一個node.

Retrieval 檢索 (常數時間)
- Array-based : Constant (independent of i)
- Pointer-based : Depends on i (線性時間).

Insertion and deletion 新增/刪除
- Array-based : Requires shifting of data 需搬移.
- Pointer-based : Requires a traversal 需走訪.

∴動第一個：陣列難、pointer易

動最後一個：陣列易、pointer難

□ Saving and Restoring a Linked List by Using a File

- Write only data to a file, not pointers 只存資料，不留指標
- Recreate the list from the file by placing each item at the end of the linked list. (Use a tail pointer)
  重建：循序加入尾端.

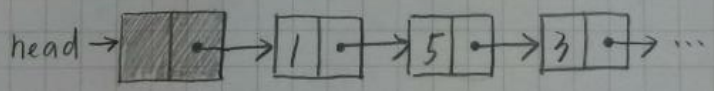
□ Variations : Circular Linked List

- Last node points to the first node 最後一個節點再指向第一個節點.
- Every node has a successor.
- No node in a circular linked list contain NULL

△會設計成環狀⇒通常是尾巴資料常需要更動

□ Variations : Dummy Head Node. (空的)

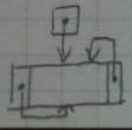
- Always present, even when the linked list is empty.
- Insertion and deletion algorithms initialize prev to point to the dummy head node, rather than to NULL.
  - Eliminates the special case



Dummy head node

□ Variations : Doubly Linked Lists (雙向)

- Each node point to both its predecessor and its successor
  - precede pointer and next pointer
  - Often has a dummy head node.
  - often circular to eliminate special cases



一、虛擬串列首的上一個指向最後節點
二、最後節點的下一個指向虛擬向列首