Recursive Functions

- Factorial 階乘
- Greatest Common Divisor 最大公因數
- Search in Array 搜尋
- Fabonacci series 費式數列
- Combinatorial numbers 組合數
- Towers of Hanoi 河內塔

* Write a string of characters in reverse order 字串長度減一

<sol> Each recursive step of the solution diminishes by 1 the length of the string to be written backward.

<Base case> empty string

```
void WriteBackward (string s, int size) {
  if (size > 0) {
    cout << s.substr (size-1, 1);     // 輸出最後一個字元
    WriteBackward (s, size-1);        // 遞迴呼叫
  } // if
                                      // size=0 is the base case
                                      // do nothing
} // WriteBackward()
```

* 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.

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

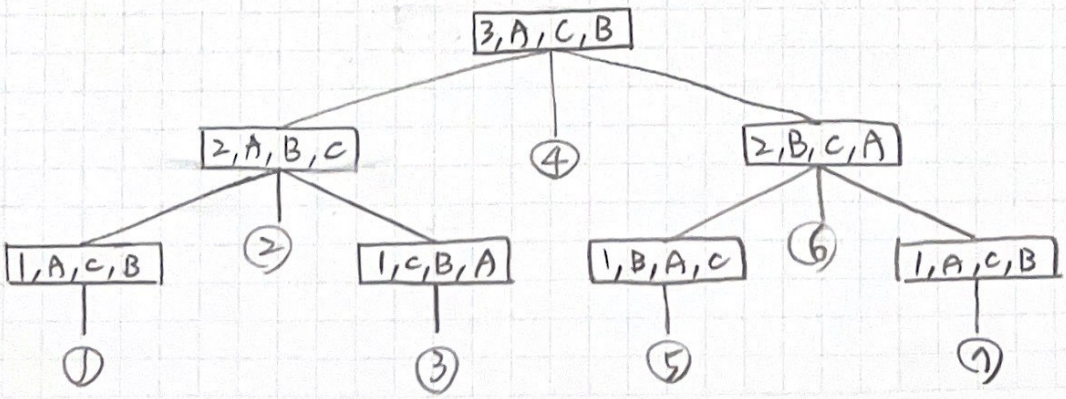
☆ Four steps

1. 遞迴定義　ex. sum(a+...b) = sum(a+b+...+1)+b
2. 問題簡化　ex. b → b+1
3. 終止條件　ex. b, b+1, ..., a ⇒ b==a
4. 保證終止　ex. natural numbers a>b

※ 河內塔

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

(個數 count, 起桌 source, 終桌 destination, 輔助 spare)



* Principles of Object-Oriented Programing

- Object-oriented languages enabled us to build class of objects (called instance)

- A class combines
  * Attributes (characteristics) 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.

Abstract Data Types : motives

※ Modularity 模組化

- Keeps the complexity of a large program manageable by systematically controlling the interaction of its components.
- Isolates errors
- Eliminates redundancies

※ A modularized program is

- Easier to write, read and modify

※ Cohesion = modules perform single well-defined tasks

- highly cohesive modules desired 高內聚

※ Coupling = measure of dependence among modules

- Loosely coupled modules desired 低耦合

※ Functional abstraction 功能性的抽象化

- Separates the purpose and use of a module from its implementatio
- A module's specifications should 描述
  - * Detail how the module behaves
  - * Be independent of module's implementation 實作

※ Information hiding 資訊隱藏

- Hides certain implement details within a module
- Makes these details inaccessible from outside the module

Abstract Data Types = concepts

※ The isolation of modules is not total

- A function's specification, or constract, governs how it interacts with other modules.

Abstract Data Types = goals

※ Typical operations on data

- Add data to a data collection
- Remove data from a data collection
- Ask questions about the data in a data collection

※ Data Abstraction 資料抽象化

- Asks you to think what you can do to a collection of data independently of how you do it.
- Allows you to develop each data structure in relative isolation from the rest of the solution
- A natural extension of functional abstraction

Abstract Data Type (ADT)

※ An ADT is composed of
- A collection of data
- A set of operations on that data

※ Specifications of an ADT indicate 描述
- What the ADT operations do, not how to implement them

※ Implement of an ADT 实作
- Includes choosing a particular data structure

和記憶體配置有關

※ C++ Classes : Constructors

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

* A class can have several constructors
  - A default constructor has no argument. 預設建構式
  - The compiler will generate a default constructor if you do not define any constructors.

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

※ C++ Classes : Destructors 解構式

- Deskorys an instance of an object when 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 don't define one

※ Inheritance in C++ 類別的繼承

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

* C++ Exceptions 例外處理

- A function can indicate that an error has occured 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 exception

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

* Array has a fixed size 陣列
  * Data must be shifte during insertions and deletions. (需移動資料)

* Linked list is able to grow in size as needed 鏈結串連 (Pointer)
  * Doesn't require the shifiting of items during insertions and deletions. (不需移動資料)

* 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; // safe guard
// 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

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];

★若new了一個新東西，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

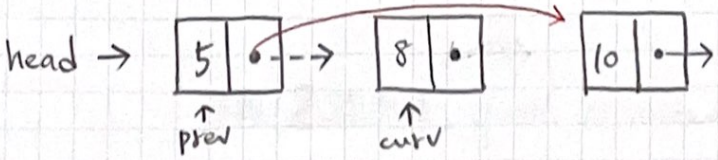
```
struct Node {
  int item;
  Node *start;
};
```

- 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.
- 鏈結串聯的好處是散落在記憶体的不同位置，可不連號，配置陣列需是連號的！

- Deleting an interior node

prev → next = cur → next / pre → next = prev → next → next




```
delete cur;
cur = NULL;
```