Chapter 1 遞迴

特點：1. 將大問題分解成更小、獨立的問題
2. 是一種包含迴圈的另類迭代

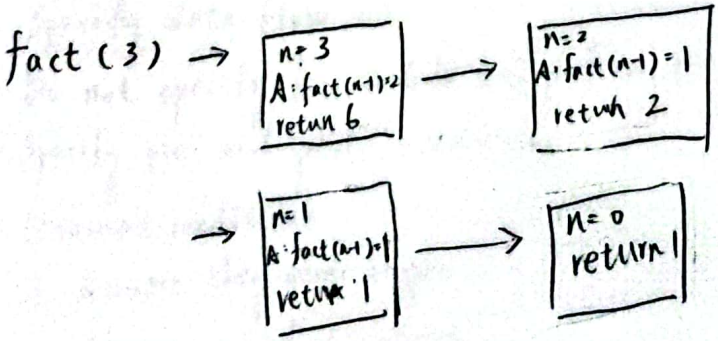
Binary Search：
1. 重複將資料對半再對半，每次尋找含有目標的那一邊
2. 採用分而擊之的策略

Facts：
1. 遞迴 function 會呼叫自己
2. 每次的呼叫都是要解決一個獨立、更小的問題
3. 中止條件 → base case
4. 在這些更小的問題中一定含有 base case，且 base case 可讓這次呼叫結束

E.g Factorial 階乘

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

Box trace：
1. 一種系統化、能夠追縱遞迴函式動作的方式
2. 每個 box 都對應一個啟動紀錄



Chapter 2

Object-Oriented Programming:

- Object-Oriented language enables us to build classes of objects.
- 一個 class 包含:
  - Attributes (屬性) of objects of a single type
    1. Typically data
    2. Called data members
  - Behaviors (運算)
    1. Typically operate on the data
    2. Called methods or member functions

3大特徵:

1. Encapsulations 封裝
   - Objects combine data and operations
   - Hide inner details
2. Inheritance 繼承
   - Classes can inherit properties from other classes
   - Existing classes can be reused
3. Polymorphism 多型
   - Objects can determine appropriate operations at execution time

Operation Contracts:

- Document the use and limitations of a method
- Specify data flow
- Do not specify how module will perform its tasks
- Specify pre- and post- conditions
- Unusual conditions:
  1. Assume they never happen
  2. Ignore invalid situations
  3. Return a value that signals a problem
  4. Throw an exception

Modularity 模組化：

- 將一個大問題系統化地分割成許多小部分來管理
- 可排除錯誤
- 減少重複性

How to achieve a Better Solution？

- Cohesion 高內聚：盡可能讓每個模組只做一件事
- Coupling 低耦合：主程式傳給模組的參數愈少愈好

Functional abstraction：

- 將模組的目的和其實作分開
- 一個模組的描述（Specifications）需要：
  1. 詳細規畫其該做的事
  2. 實作和描述各自獨立

Information hiding：

- 將特定實作隱藏起來
- 讓被隱藏的部分無法從外面修改
- 修改這些資訊的時候不影響其他模組運作

Data abstraction：

- 想好你對資料要做什麼而不是如何做
- 能夠分開獨立開發每個資料結構
- 是functional abstraction的自然延伸

Chapter 3:

Pointers:

- A pointer contains the location, or adress in memory, of a memory cell
  - Declaration of an int pointer variable p:

    ```
    int *p;
    ```

    - Initially undefined, but not NULL
    - Static allocation
- *p represents the memory cells to which p points
- To place the address of a variable into a pointer variable, use:

  ```
  p = &x;
  p = new int;
  ```

  - Dynamic allocation of a memory cell that can contain an integer
  - If new cannot allocate memory, it throws the exception std::bad_alloc
- The delete operator returns dynamically allocated memory to the system for reuse, and leaves the variable content undefined

  ```
  delet p;
  ```

  - A pointer to a deallocated memory (*p) cell is possible and dangerous

  ```
  p = NULL; // safeguard
  ```

Dynamic Allocation of Arrays:

- The new operator can be used to allocate an array dynamically

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

- An array name is a pointer to its first element

  ```
  anArray[2] = *(anArray + 2)
  ```

- The size of a dynamically allocated array can be increased

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

- After increasing, data stored in the old array should be moved to the new array

Chapter 4:

The basics of grammers:

- A language
  - A set of strings of symbol
  - Eg. C++ · English
- A grammer:
  - 形成一語言的 string 的規則
  - Eg. English grammer, C++ syntax rules

Defining language:

- If a C++ program is one long string of characters, the language of C++ program is defined as:

  C++ Programs = { string w : w is syntactically correct C++ program }

- A language doesn't have to be a programming or a communication language.

  Eg. AlgebraicExpressions = { string w: w is an algebraic expression }

Symbols used in grammers:

- x|y → x or y
- xy . x·y → x followed by y
- <word> means any instance of word that the definitions define.
- E.g. <number> = <number><digit> | <digit>
  <digit> = 0 | 1 | 2 | ... | 9

  <addition> = <addition> + <digit> | <digit>
  <digit> = 0 | 1 | 2 | ... | 9
- A C++ identifier begins with a letter and is followed by 0 or more letters and digits
- Language of C++ identifiers: C++Ids = { w : w is a legal C++ identifier }
- Grammer:

  <identifier> = <letter> | <identifier><letter> | <identifier><digit>

  <letter> = a | b | c | ... | A | B | C | ... | Z

  <digit> = 0 | 1 | 2 | ... | 8 | 9

- A recognition algorithm sees whether a given string is in the language.

```
isId (in w string) : boolean
 if (w is of length 1)
   if (w is a letter) return true;
   else return false;
 else if (the last character of w is a letter | a digit)
    return isId (w minus its last character);
 else return false;
```

Algebraic Expressions

- Infix Expressions (中序)
  - 運算子出現在運算元之間 e.g. a+b
- Prefix Expressions (前序)
  - 運算子出現在運算元之前 e.g. +ab
- Postfix Expressions (後序)
  - 運算子出現在運算元之後 e.g. ab+

Fully Parenthesized Expressions (完整括號):

- Fully parenthesized infix expressions:
  - Do not require precedence rules or rules for association
  - Not convenient for programmers
- Grammer:

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

  <operator> = + | - | * | /

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

Infix to Prefix:

((a+b)*c) → *+abc (+ : 2, * : 2)

Infix to Postfix

((a+b)*c) → ab+c* (+ : 2, * : 2)

Advantages of Prefix and Postfix:

1. 無優先權 (precedence rules)
2. 無結合律 (association rules)
3. 無括號 (parentheses)
4. Simple grammers
5. Straightforward recognition and evaluation algorithms