10-類別(class)

10-1 類別(class)與物件(object)

一、物件（object）與類別（class）

在 C++ 中物件和類別有很嚴謹的定義，我們在這裡僅使用簡單例子來做介紹。

什麼是物件(Object)？你的iPhone是個物件、你現在坐著的這張椅子是個物件，小狗小黃也是個物件。

那類別呢(Class)？你的iPhone是個物件，隔壁同學的HTC One也是個物件，而它們同屬於「手機」這個類別。小黃是個物件、小黑也是個物件，牠們同屬於「狗」這個類別。

二、建立類別與產生物件

在產生物件之前必須先建立類別，在C++中建立類別要使用到 class 這個關鍵字，一個最簡單的類別長這個模樣(注意：最後有一個分號)。

```
class 類別名稱 {
};
```

假設我們要建立一個dog類別，可以這樣寫。

```
class dog {
};
```

接下來我們可以這樣產生兩個 dog 類別的物件：shiro（小白） 和 kuro（小黑）。

```
dog shiro;
dog kuro;
```

或

```
dog shiro, kuro;
```

就像在定義 int 或 float 等原生型別變數一樣。

三、屬性 (attribute)/資料成員(data member)

前面這個 dog類別是一個很單純的類別，它產生的兩個物件也沒什麼用處。現在我們要給它一點變化，讓狗有不同的顏色和叫聲。

```
class dog {
  public:
    string color;
    string sound;
};
```

public 關鍵字表示下面出現的屬性都是「公開的」，你可以在程式的任何地方來改變其值。

在這個例子中我們有兩個 string 型別的屬性 color 和 sound。我們可以用 `物件名稱.屬性名稱` 的型式來設定或取用特定物件的某個屬性值。

```
dog shiro;
dog kuro;

shiro.color = "白色";
kuro.color = "黑色";

cout << "shiro 的顏色是" << shiro.color << endl;
cout << "kuro 的顏色是" << kuro.color << endl;
```

四、成員函數(member function)

到目前為止，你會覺得 class 和之前學的 struct 一樣。

但是除了屬性之外，我們還可以在類別中加入成員函數，讓該類別的物件可以「做些事情」，例如：我們可以在 dog 類別中加入 bark 這個成員函數，讓小白和小黑可以叫。

```
class dog {
  public:
    string color;
    string sound;

    void bark() {
        cout << sound << endl;
    }
};

int main()
{
    dog shiro, kuro;

    shiro.color = "白色";
    shiro.sound = "汪！";
    kuro.color = "黑色";
    kuro.sound = "汪汪汪！";

    cout << "shiro 咬他！" << endl;
    shiro.bark();                    // 汪！
    cout << "kuro 咬他！" << endl;
    kuro.bark();                     // 汪汪汪！

    return 0;
}
```

五、特殊的成員函數- 建構(constructor), 解構(destructor)

constructor 是一個和類別同名的成員函數，在一個物件被產生時會自動被呼叫，而且 `沒有傳回值`

```
class dog {
  public:
    string color;
    string sound;

    void bark() {
        cout << sound << endl;;
    }
    dog() {
        cout << "有一隻小狗誕生了！" << endl;
    }
};

int main()
{
    dog shiro, kuro;

    return 0;
}
```

一個類別可以有很多個 constructor，每個 constructor 的參數列都必須不一樣，在產生物件時，根據你使用參數的不同，相對應的 constructor 會自動被呼叫。

沒有參數的 constructor 稱為 `default constructor`。

```
class dog {
  public:
    string color;
    string sound;

    void bark() {
        cout << sound << endl;;
    }
    dog() {
        cout << "有一隻小狗誕生了！" << endl;
    }
    dog(string _color, string _sound) {
        color = _color;
        sound = _sound;
        cout << "有一隻" << color << "小狗誕生了！" << sound << endl;
    }
};

int main()
{
    dog kuro;
    dog shiro("白色","汪！");

    return 0;
}
```

destructor 是物件「死亡」時會被自動呼叫的成員函數，每個類別只有一個 destructor，它的名字是 ~類別名稱，**`destructor 沒有參數也沒有傳回值`**。

```
class dog {
  public:
    string color;
    string sound;
    string name;

    void bark() {
        cout << sound << endl;;
    }

    dog(string _name, string _color, string _sound) {
        name = _name;
        color = _color;
        sound = _sound;
        cout << "有一隻名為 " << name << " 的 " << color << " 小狗誕生了！"
                << sound << endl;
    }
    ~dog() {
        cout << name << " 上天堂了。" << endl;
    }
};

int main()
{
    dog shiro("小白", "白色", "汪！");
    dog *kuro = new dog("小黑", "黑色","喵！");

    delete kuro;

    return 0;
}
```

六、使用指標操作物件

使用指標操作物件與使用指標操作 struct 變數一樣，必須用到「->」。

```
int main()
```

```
{
    dog *kuro = new dog("小黑", "黑色","喵！");

    kuro->sound = "汪！汪！";
    kuro->bark();

    delete kuro;

    return 0;
}
```

七、this 指標

this 是一個指向「自己這個物件」的指標。

成員函數的參數名稱和屬性名稱完全一樣時，不加 this-> 編譯器會把兩邊都當作參數，屬性永遠不會被賦值。

```
#include <iostream>

using namespace std;

class Student {
public:
    string name;
    int age;

    void setName(string name) {
        //name = name;         // 自己賦值給自己，成員變數沒有被改到
        this->name = name;     // 左邊是成員變數，右邊是參數
    }
    void setAge(int age) {
        //age = age;           // 自己賦值給自己，成員變數沒有被改到
        this->age = age;       // 同上
    }

    void printInfo() {
        cout << "Name: " << name << ", Age: " << age << endl;
    }
};


int main()
{
    Student std;
    std.setName("Alice");
    std.setAge(20);

    std.printInfo();

    return 0;
}
```

使用 this

```
Name: Alice, Age: 20
```

不使用 this

```
Name: , Age: -103249728
```

10-2 存取控制——public 與 private

一、class 裡的東西，不是誰都可以動的

先看一個銀行帳戶的例子。

```
#include <iostream>

using namespace std;

class BankAccount {
public:
    string owner;
    int balance;
};

int main() {
    BankAccount acc;
    acc.owner   = "小明";
    acc.balance = 1000;

    acc.balance = -99999;   // 沒有任何阻擋，帳戶餘額變成負的！

    cout << acc.balance << endl;
}
```

這是因為我們之前在 class 裡用了 `public:` 這個存取修飾詞。 public 的意思就是「公開的」，誰想看、誰想改都可以。

用 `private:` 將存取限制改為「私有的」看看差別。

```
#include <iostream>

using namespace std;

class BankAccount {
private:
    string owner;
    int balance;
};

int main() {
    BankAccount acc;
    acc.owner   = "小明";
    acc.balance = 1000;

    acc.balance = -99999;   // 沒有任何阻擋，帳戶餘額變成負的！

    cout << acc.balance << endl;
}
```

修改之後，連編譯都失敗，出現了大量如下的錯誤訊息。

因為使用 privte 修飾的屬性和成員函數，在該 class 之外(第 5~9 行之外)都無法存取，在編譯時就攔下了不合法的存取。

```
main.cpp:13:9: error: 'std::string BankAccount::owner' is private within this context
   13 |     acc.owner   = "小明";
      |         ^~~~~
main.cpp:7:12: note: declared private here
    7 |     string owner;
      |            ^~~~~
main.cpp:14:9: error: 'int BankAccount::balance' is private within this context
   14 |     acc.balance = 1000;
```

```
      |         ^~~~~~
main.cpp:8:9: note: declared private here
    8 |     int balance;
      |         ^~~~~~
main.cpp:16:9: error: 'int BankAccount::balance' is private within this context
   16 |     acc.balance = -99999;   // 沒有任何阻擋，帳戶餘額變成負的！
      |         ^~~~~~
main.cpp:8:9: note: declared private here
    8 |     int balance;
      |         ^~~~~~
main.cpp:18:17: error: 'int BankAccount::balance' is private within this context
   18 |     cout << acc.balance << endl;
      |                 ^~~~~~~
main.cpp:8:9: note: declared private here
    8 |     int balance;
      |         ^~~~~~
```

二、提供外界有限的操作

設成 public 太危險，設成 private 又動不了，怎麼辦呢?

我們可以

- 把內部屬性設為 private
- 用 public 開放幾個成員函數，給外界有限能力操作內部屬性

```
#include <iostream>

using namespace std;

class BankAccount {
private:
    string owner;
    int balance;    // 外部無法直接存取

public:
    // 以下成員函數，開放給外部使用，提供存取 balance 的管道
    bool withdraw(int amount) {
        if (amount > balance) {
            cout << "餘額不足！" << endl;
            return false;
        }
        balance -= amount;
        return true;
    }

    void deposit(int amount) {
        if (amount <= 0) {
            cout << "存款金額必須大於 0！" << endl;
            return;
        }
        balance += amount;
    }

    int getBalance() { return balance; }
};

int main()
{
    BankAccount account;
    account.deposit(1000);  // 存款 1000 元
    cout << "目前餘額: " << account.getBalance() << " 元" << endl;

    if (account.withdraw(500)) {  // 提款 500 元
        cout << "提款成功！" << endl;
    }
    cout << "目前餘額: " << account.getBalance() << " 元" << endl;
```

```
    if (!account.withdraw(600)) {  // 嘗試提款 600 元，應該會失敗
        cout << "提款失敗！" << endl;
    }
    cout << "目前餘額: " << account.getBalance() << " 元" << endl;

    return 0;
}
```

現在外部程式只能透過 `withdraw()` 和 `deposit()` 來改變餘額，透過 `getBalance()` 來取得餘額，非法操作會被擋下來。

public、private 存取修飾詞的作用範圍

```
class Example {
// 這裡沒寫 → class 預設是 private
    int secret;

public:
    int visible;
    void doSomething() { }

private:
    int alsoSecret;
};
```

`public:` 和 `private:` 後面的所有成員，都套用該修飾子，直到遇到下一個修飾子為止。

三、封裝(encapsulate)的概念

有時候我們不希望 class 的使用者直接去更動某些 data member 的資料，這時我們會將這些 data member 宣告在 private 區域中，並在 public 區域中提供 member function 來操作這些 data member。

以手錶為例，一般來說製表師傅不會希望使用者自己去撥動錶內的齒輪，所以他會把手錶內部的複雜機構用錶殼緊緊的封裝起來，不過他還是會提供一些按鈕或旋鈕來讓我們調整時間。

- private – 手錶的內部複雜機構
- public – 調整時間的按鈕或旋鈕

當我們試著在非成員函數裡存取 private 的 data member 或 member function 時，在編譯時期就會出現錯誤訊息。

這就是「物件導向程式設計」中很重要的「封裝」概念。我們設計了一個很精巧的 class，包含了很多的屬性和成員函數，但是對外界來說，它被一個「盒子」封裝起來，你只操作這個盒子外漏的那幾個界面。

10-3 如何建立複雜的類別

一、成員函數可以定義在 class 之外

我們可以只在 class body 中 `宣告` member function，再將它 `定義` 在 class body 之外。

```
class sprite {
private:
    string name;
public:
    sprite(string _name);  // 只有宣告
    string getName();      // 只有宣告
};

// 定義 sprite 類別的 sprite 成員函數
sprite::sprite(string _name)
{
    name = _name;
}

// 定義 sprite 類別的 getName 成員函數
string sprite::getName()
{
    return name;
}
```

以下是一個遊戲中角色的範例

```
#include <iostream>

using namespace std;

class sprite
{
private:
    string name;
    string status;
    int HP;
    int maxHP;

    void setStatus(void);  // 根據 hp 設定健康狀態
    void drawHpBar(void);  // 繪製 hp 長條圖
public:
    sprite(string _name);
    string getName();      // 取得姓名
    void decHP(int n);     // 減少 HP
    void addHP(int n);     // 增加 HP
    void show(void);       // 顯示狀態
};

void sprite::setStatus(void)
{
    if(HP >= maxHP *0.9)
        status = "健康";
    else if(HP >= maxHP *0.6)
        status = "受傷";
    else if(HP >= maxHP *0.3)
        status = "重傷";
    else if(HP > 0)
        status = "瀕死";
    else
        status = "死亡";
}

sprite::sprite(string _name)
```

```
{
    name = _name;
    HP = maxHP = 20;
    status = "健康";
}

string sprite::getName()
{
    return name;
}

void sprite::decHP(int n)
{
    HP = HP - n;
    if(HP <0)
        HP = 0;
    setStatus();
}

void sprite::addHP(int n)
{
    HP = HP + n;
    if(HP > maxHP)
        HP = maxHP;
    setStatus();
}

void sprite::drawHpBar(void)
{
    int a = 10*HP/maxHP;
    cout << "[";
    for(int i=0; i<a; i++)
        cout << "#";
    for(int i=a; i<10; i++)
        cout << "-";
     cout << "] ";
}

void sprite::show(void)
{
    cout << "[" << name << "] ";
    cout << "狀態：" << status << endl;
    drawHpBar();
    cout << "HP: " << HP  << "/" << maxHP << endl;
    cout << endl;
}

void attack(sprite &s, int n)
{
    cout << s.getName() << " 受到 " << n << " 點的傷害。" << endl;
    cout << endl;
    s.decHP(n);
}


int main()
{
    sprite fighter1("David");
    fighter1.show();
    attack(fighter1, 6);
    fighter1.show();
    attack(fighter1, 6);
    fighter1.show();
    attack(fighter1, 6);
    fighter1.show();
    attack(fighter1, 6);
    fighter1.show();
```

```
    return 0;
}
```

```
David 受到 6 點的傷害。
[David] 狀態：受傷
[#######---] HP: 14/20


David 受到 6 點的傷害。


[David] 狀態：重傷
[####------] HP: 8/20


David 受到 6 點的傷害。


[David] 狀態：瀕死
[#---------] HP: 2/20


David 受到 6 點的傷害。


[David] 狀態：死亡
[----------] HP: 0/20
```

二、將類別放在獨立的檔案中

在前一個例子裡，我們把 sprite 類別和 main 放在同一個檔案裡，在程式碼短的情形下是沒問題的。但在動輒使用到數十乃至上百個類別的程式裡，這麼做就不切實際了，會造成維護和分工上的困擾。

我們可以將sprite類別移到獨立的檔案裡，這樣修改 sprite 時就可以專注在 sprite 的程式碼，main.cpp 裡也不會顯得凌亂，建置專案時的效率也會更好。

sprite.h

```
#ifndef SPRITE_H_INCLUDED
#define SPRITE_H_INCLUDED

#include <iostream>

using namespace std;

class sprite
{
private:
    string name;
    string status;
    int hp;
    int maxHp;

    void setStatus(void);  // 根據 hp 設定健康狀態
    void drawHpBar(void);  // 繪製 hp 長條圖
public:
    sprite(string _name);
    string getName();      // 取得姓名
    void decHP(int n);     // 減少 HP
    void addHP(int n);     // 增加 HP
    void show(void);       // 顯示狀態
};

#endif // SPRITE_H_INCLUDED
```

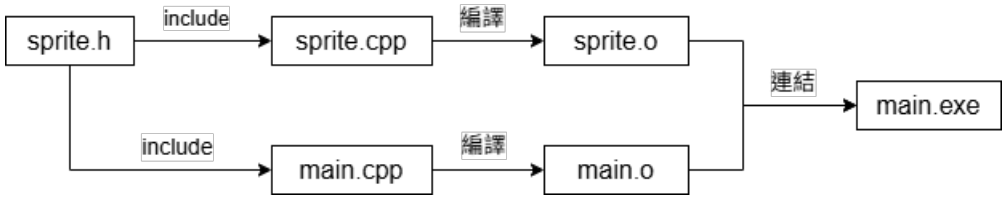
sprite.cpp

```
#include "sprite.h"

void sprite::setStatus(void)
{
    if(HP >= maxHP*0.9)
        status = "健康";
    else if(HP >= maxHP*0.6)
        status = "受傷";
    else if(HP >= maxHP*0.3)
        status = "重傷";
    else if(HP > 0)
        status = "瀕死";
    else
        status = "死亡";
}

sprite::sprite(string _name)
{
    name = _name;
    HP = maxHP = 20;
    status = "健康";
}

string sprite::getName()
{
    return name;
}

void sprite::decHP(int n)
{
    HP = HP - n;
    if(HP<0)
        HP = 0;
    setStatus();
}

void sprite::addHP(int n)
{
    HP = HP + n;
    if(HP > maxHP)
        HP = maxHP;
    setStatus();
}

void sprite::drawHpBar(void)
{
    int a = 10*HP/maxHP;
    cout << "[";
    for(int i=0; i<a; i++)
        cout << "#";
    for(int i=a; i<10; i++)
        cout << "-";
     cout << "] ";
}

void sprite::show(void)
{
    cout << "[" << name << "] ";
    cout << "狀態：" << status << endl;
    drawHpBar();
    cout << "HP: " << HP  << "/" << maxHP << endl;
    cout << endl;
}
```

main.cpp

```
#include <iostream>
#include "sprite.h"

using namespace std;

void attack(sprite &s, int n)
{
    cout << s.getName() << " 受到 " << n << " 點的傷害。" << endl;
    cout << endl;
    s.decHP(n);
}

int main()
{
    sprite fighter1("David");
    fighter1.show();
    attack(fighter1, 6);
    fighter1.show();
    attack(fighter1, 6);
    fighter1.show();
    attack(fighter1, 6);
    fighter1.show();
    attack(fighter1, 6);
    fighter1.show();

    return 0;
}
```

建置時的檔案相依性



10-4 Class 練習題

練習一：電影票券（`Ticket`）

情境說明

你正在設計一套電影院售票系統。每張票券記錄了電影名稱、座位號碼、票價，以及是否已被使用。票券一旦使用就不能再次入場；票價不能設為負數。

規格列表

成員變數（皆為 `private`）

變數名稱	型態	說明
`movieName`	`string`	電影名稱
`seatNumber`	`int`	座位號碼
`price`	`int`	票價（元）
`used`	`bool`	是否已使用

建構子

建構子	初始值
`Ticket()`	`movieName="未命名"`、`seatNumber=0`、`price=0`、`used=false`
`Ticket(string movieName, int seatNumber, int price)`	依參數設定，`used=false`

> “兩個建構子的參數名稱與成員變數相同，需使用 `this->` 或初始化列表。

Getter（皆加 `const`）

函式	回傳型態	說明
`getMovieName()`	`string`	回傳電影名稱
`getSeatNumber()`	`int`	回傳座位號碼
`getPrice()`	`int`	回傳票價
`isUsed()`	`bool`	回傳是否已使用

Setter（含驗證）

函式	驗證規則
`setPrice(int price)`	`price < 0` 時印出 `"票價不能為負數！"` 並放棄修改

其他成員函式

函式	回傳	說明
`use()`	`bool`	若已使用，印出 `"此票券已使用過！"` 並回傳 `false`；否則將 `used` 設為 `true`，印出入場訊息並回傳 `true`
`print() const`	`void`	印出票券所有資訊（格式見預期輸出）

`main()`

```
int main() {
    Ticket t1("星際大戰", 12, 280);
    Ticket t2;
```

```
    t1.print();
    t2.print();

    t1.use();
    t1.use();           // 重複使用

    t1.setPrice(-100);  // 非法
    t1.setPrice(250);   // 合法

    cout << t1.getMovieName() << " 目前票價：" << t1.getPrice() << " 元" << endl;

    t1.print();
}
```

預期輸出

```
[票券] 星際大戰 | 座位 12 | 票價 280 元 | 狀態：未使用
[票券] 未命名 | 座位 0 | 票價 0 元 | 狀態：未使用
入場成功！電影：星際大戰，座位：12
此票券已使用過！
票價不能為負數！
星際大戰 目前票價：250 元
[票券] 星際大戰 | 座位 12 | 票價 250 元 | 狀態：已使用
```

練習二：水壺（`Pitcher`）

情境說明

你設計一個登山水壺管理程式。每個水壺有名稱、最大容量和目前水量。注水時若超過容量會溢出；倒水時若水量不足則失敗；容量不能設得比目前水量還小。

規格列表

成員變數（皆為 `private`）

變數名稱	型態	說明
`name`	`string`	水壺名稱
`capacity`	`int`	最大容量（ml）
`current`	`int`	目前水量（ml）

建構子

建構子	初始值
`Pitcher()`	`name="水壺"`、`capacity=1000`、`current=0`
`Pitcher(string name, int capacity)`	依參數設定，`current=0`

Getter（皆加 `const`）

函式	回傳型態	說明
`getName()`	`string`	回傳水壺名稱
`getCapacity()`	`int`	回傳最大容量
`getCurrent()`	`int`	回傳目前水量

Setter（含驗證）

函式	驗證規則
`setCapacity(int capacity)`	`≤ 0` 印出 `"容量必須大於 0！"`；新容量 `< current` 印出 `"新容量小於目前水量，無法設定！"`

其他成員函式

函式	回傳	說明
`fill(int amount)`	`void`	注水；若超出容量則補滿並印出溢出量，否則正常注水
`pour(int amount)`	`bool`	倒水；水量不足回傳 `false` 並印出提示，否則正常倒出回傳 `true`
`status() const`	`void`	印出水壺狀態，含百分比（格式見預期輸出）

`main()`

```
int main() {
    Pitcher p1("運動水壺", 750);
    Pitcher p2;

    p1.status();
    p2.status();

    p1.fill(500);
    p1.fill(400);       // 會溢出

    p1.pour(200);
    p1.pour(600);       // 水量不足

    p1.setCapacity(-1); // 非法
    p1.setCapacity(100);// 非法（小於目前水量 550）

    cout << p1.getName() << " 容量：" << p1.getCapacity() << " ml" << endl;

    p1.status();
}
```

預期輸出

```
[運動水壺] 0 / 750 ml ( 0% )
[水壺] 0 / 1000 ml ( 0% )
運動水壺 注入 500 ml。
運動水壺 已裝滿，多餘 150 ml 溢出。
運動水壺 倒出 200 ml。
運動水壺 水量不足，無法倒出 600 ml！
容量必須大於 0！
新容量小於目前水量，無法設定！
運動水壺 容量：750 ml
[運動水壺] 550 / 750 ml ( 73% )
```

練習三：遊戲計分板（`ScoreBoard`）

情境說明

你正在設計一款小遊戲的計分系統。計分板記錄玩家名稱、歷史最高分、當前分數和剩餘生命數。每次得分若超過歷史最高分則更新紀錄；失去所有生命時宣告遊戲結束；重新開始時分數歸零但最高分保留。

規格列表

成員變數（皆為 `private`）

變數名稱	型態	說明
`playerName`	`string`	玩家名稱
`highScore`	`int`	歷史最高分
`currentScore`	`int`	當前分數
`lives`	`int`	剩餘生命數

建構子

建構子	初始值
`ScoreBoard()`	`playerName="玩家"`、`highScore=0`、`currentScore=0`、`lives=3`
`ScoreBoard(string playerName, int lives)`	依參數設定，`highScore=0`、`currentScore=0`

Getter（皆加 `const`）

函式	回傳型態	說明
`getPlayerName()`	`string`	回傳玩家名稱
`getHighScore()`	`int`	回傳歷史最高分
`getCurrentScore()`	`int`	回傳當前分數
`getLives()`	`int`	回傳剩餘生命數

Setter（含驗證）

函式	驗證規則
`setPlayerName(string playerName)`	名稱為空字串時印出 `"玩家名稱不能為空！"` 並放棄修改

其他成員函式

函式	回傳	說明
`addScore(int points)`	`void`	加分（`≤0` 印出提示拒絕）；若 `currentScore > highScore` 則更新並印出新紀錄訊息
`loseLife()`	`bool`	扣一條命；`lives=0` 時印出遊戲結束訊息並回傳 `false`，否則回傳 `true`
`reset()`	`void`	`currentScore` 歸零、`lives` 回復 3，`highScore` 保留，印出提示
`print() const`	`void`	印出所有資訊（格式見預期輸出）

`main()`

```
int main() {
    ScoreBoard sb("小明", 3);
    ScoreBoard sb2;

    sb.print();
    sb2.print();

    sb.addScore(100);
    sb.addScore(250);
    sb.addScore(50);

    sb.loseLife();
    sb.loseLife();
    sb.loseLife();      // 第三條命，遊戲結束

    sb.setPlayerName(""); // 非法
    sb.setPlayerName("小明Pro");

    sb.reset();
    sb.print();

    sb.addScore(500);   // 超越舊最高分
    sb.print();
}
```

預期輸出

```
== 小明 == 當前：0 | 最高：0 | 命：3
== 玩家 == 當前：0 | 最高：0 | 命：3
新紀錄！最高分更新為 100
小明 得 100 分，目前：100
新紀錄！最高分更新為 350
小明 得 250 分，目前：350
新紀錄！最高分更新為 400
```

```
小明 得 50 分，目前：400
小明 失去一條命，剩餘：2
小明 失去一條命，剩餘：1
小明 失去一條命，剩餘：0
遊戲結束！小明 最終得分：400
玩家名稱不能為空！
小明Pro 重新開始，最高分保留：400
== 小明Pro == 當前：0 | 最高：400 | 命：3
新紀錄！最高分更新為 500
小明Pro 得 500 分，目前：500
== 小明Pro == 當前：500 | 最高：500 | 命：3
```