Chapter 6

Constructors and Other Tools

想一想

- 遊戲中有哪些類別，遊戲中又會用到這些類別的那些資訊?
- 又有哪些資訊是遊戲中通用的?

遊戲中有主角、機關、生物等各式類別

想一想

- 遊戲中有哪些類別，遊戲中又會用到這些類別的那些資訊?
  - 英雄: 血量、位置、攻擊力、…。
  - 生物: 血量、行為類型、…。
  - …。
- 又有哪些資訊是遊戲中通用的?
  - 同一種類生物數量、樓層數、…。

Learning Objectives

- Constructors
  - Definitions
  - Calling
- More Tools
  - `const` parameter modifier
  - `inline` functions
  - `static` member data
- Vectors
  - Introduction to `vector` class

Constructors (建構子)

- Initialization of objects
  - Initialize some or all **member** variables
  - Other actions possible as well
- A special kind of member functions
  - Automatically called when object declared
- Constructors defined like any member function except:
  1. Must have **same name** as class
  2. **Cannot return** a value; **not** even **void**!

Constructor Definition Example

- Class definition with constructor:
  -
```
class Position
{
public:
    //Initializes x and y to arguments.
    Position(int x, int y);
    // Initializes y to 0 of given x.
    Position(int x);
    //Initializes the position to (0,0).
    Position( );
    void input();
    void output();
    …
private:
    int x = 0;
    int y = 0;
}
```

Constructor Notes

- Notice name of constructor: `Position`
  - **Same name** as class itself!
- Constructor declaration has **no** return-type
  - Not even void!
- Constructor in **public** section
  - It's called when objects are declared
  - If private, could not declare(or instantiate) objects normally
    - Sometimes we intend to make the constructor private!

Constructor Code

- Constructor definition is like all other member functions:

```
Position::Position(int xPosition, int yPosition){
    x = xPosition;
    y = yPosition;
}
```

- Note same name around `::`
  - Clearly identifies a constructor
- Note no return type, just as in class declaration

Alternative Definition

- Previous definition equivalent to:

```
Position::Position(  int xPos,
                     int yPos)
          : x(xPosition), y(yPosition)
{…}
```

- Third line called "**Initialization Section**"
- Body left empty
- Preferable definition version
  - Variables are initialized through constructor instead of assignment

Calling

- Declare objects:

```
Position heroPos(7,4), creaturePos(5,5);
```

- Objects are created here
  - Constructor is called
  - Values in parentheses passed as arguments to constructor
  - Member variables initialized:

```
heroPos.x = 7        creaturePos.x = 5
heroPos.y = 4        creaturePos.y = 5
```

- Consider:
  -

```
Position heroPos, creaturePos;
heroPos.Position(7, 4);   // ILLEGAL!
creaturePos.Position(5, 5);     // ILLEGAL!
```

- Seemingly OK…
  - CANNOT call constructors like other member functions!

Constructor Additional Purpose

- Not just initialize data
- Body doesn't have to be empty
  - In initializer version
- Validate the data!
  - Ensure only appropriate data is assigned to class private member variables
  - Powerful OOP principle

Overloaded Constructors

- Can overload constructors just like other functions
- Recall: a signature consists of:
  - Name of function
  - Parameter list
- Provide constructors for all possible argument-lists
  - Particularly "how many"

Example

```
1)   class DayOfYear
2)   {
3)   public:
4)       DayOfYear(int monthValue, int dayValue);
5)       //Initializes the month and day to arguments.
6)       DayOfYear(int monthValue);
7)       //Initializes the date to the first of the given month.
8)       DayOfYear( );
9)       //Initializes the date to January 1.
10)      void output( );
11)  private:
12)      int month;
13)      int day;
14)  };
```

Example

```
1)   DayOfYear::DayOfYear(int monthValue, int dayValue)
2)                             : month(monthValue), day(dayValue)
3)   {
4)       testDate( );
5)   }
6)   DayOfYear::DayOfYear(int monthValue) :
                          month(monthValue), day(1)
7)   {
8)       testDate( );
9)   }
10)  DayOfYear::DayOfYear( ) : month(1), day(1)
11)  { /*Body intentionally empty.*/ }
```

Example

```
1)  void DayOfYear::output( )
2)  {
3)      switch (month)
4)      {
5)          case 1:
6)              cout << "January "; break;
7)          case 2:
8)              cout << "February "; break
9)          case 3:
10)             cout << "March "; break;
11)         case 4:
12)             cout << "April "; break;
13)         case 5:
14)             cout << "May "; break;
15)         case 6:
16)             cout << "June "; break;
17)         case 7:
18)             cout << "July "; break;
19)         case 8:
20)             cout << "August "; break;
21)         case 9:
22)             cout << "September ";
    break;
23)         case 10:
24)             cout << "October "; break;
25)         case 11:
26)             cout << "November "; break;
27)         case 12:
28)             cout << "December "; break;
29)         default:
30)             cout << "Error in
    DayOfYear::output. Contact software
    vendor.";
31)     }
32)     cout << day;
33) }
```

Example

```
1)   #include <iostream>
2)   #include <cstdlib> //for exit
3)   using namespace std;
4)   int main( )
5)   {
6)       DayOfYear date1(2, 21), date2(5), date3;
7)       cout << "Initialized dates:\n";
8)       date1.output( ); cout << endl;
9)       date2.output( ); cout << endl;
10)      date3.output( ); cout << endl;
11)      date1 = DayOfYear(10, 31);
12)      cout << "date1 reset to the following:\n";
13)      date1.output( ); cout << endl;
14)      return 0;
15)  }
```

Example: Creature

- 建立生物類別時需要產生基本的狀態資訊
  - 初始位置、血量、表示符號、...。
  - 這些狀態資訊可透過constructor 在建立類別時賦予。

遊戲中的生物有著許多資訊

Example: Creature

```
10  class Creature {//creature class
11  private:
12      ////////////
13      //主要區域
14      //creature 欄位介紹
15      ////////////
16      int health = 1;      //血量，小於零時代表死亡
17      Position pos;        //位置，表示在版面中的相對位置
18      string icon = "C"; //符號，版面中代表的符號
19      string halfIcon = "C";  //半形符號，處於特殊狀態用的符號。
20      string fullIcon = "C"; //全形符號，處於一般狀態用的符號。
21      int range = 1;           //攻擊範圍，當玩家與其的距離小於該數值會攻擊玩家
22      bool isDead = false;     //死亡狀態，血量小於零時為true
```

實作生物類別所需的基本欄位

Example: Creature

```
23      public:
24          ////////////
25          //主要區域
26          //creature construct實作
27          ////////////
28          Creature();
29          Creature(int x, int y);
30          Creature(int x, int y, string fullIcon, string halfIcon);
```

生物類別建構子

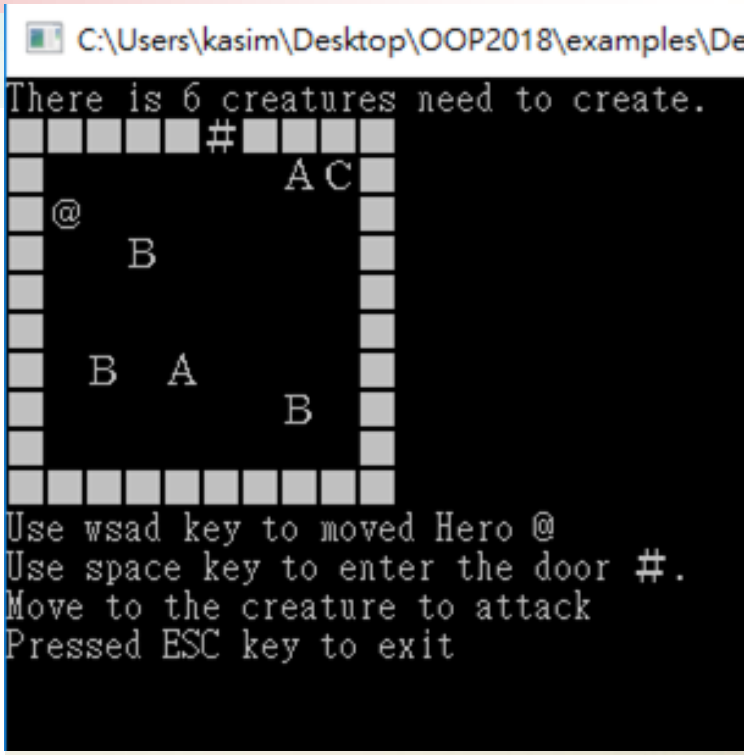
Example: Creature

```
57  ////////////
58  //主要區域
59  //creature construct實作
60  //有基本、位置、符號三種
61  ////////////
62  Creature::Creature() {};
63  Creature::Creature(int x, int y) { this->pos.x = x; this->pos.y = y; this->icon = this->fullIcon; };
64  Creature::Creature(int x, int y, string fullIcon,string halfIcon) {
65      this->pos = Position{ x,y };
66      this->fullIcon = fullIcon;
67      this->halfIcon = halfIcon;
68      this->icon = this->fullIcon;
69  }
```

生物類別建構子:將輸入資訊賦予對應欄位資料

Example: Same Creatures

```
int type = rand() % 3;
switch (type)
{
case 0:
    creatures[i] = Creature(x, y);  //create a creature at (x, y)
    break;
case 1:
    creatures[i] = Creature(x, y,"A","a");
    break;
case 2:
    creatures[i] = Creature(x, y, "B", "B");
    break;
default:
    break;

}
```



迷宮中的生物:
透過不同的建構子參數產生不同外觀

Default Constructor

- Defined as: constructor w/ no arguments
- Confusing while standard functions with no arguments:

```
callFunction(); // Including empty parentheses
```

- Object declarations with no "initializers":
  - `Position heroPos;   // This way!`
  - `Position heroPos(); // NO!`
    - Compiler sees a function declaration/prototype!
- One should always be defined
- Auto-Generated?
  - Yes & No
  - If no constructors AT ALL are defined → Yes
  - If any constructors are defined → No
- If no default constructor:
  - Cannot declare: `MyClass myObject;`
    - With no initializers

Default Constructor

- 可以明確指定要使用 Default Constructor

```
class Position{
public:
    Position() = default;
};
```

- 也可明確指定不要

```
class Position{
public:
    Position() = delete;
};
```

- Behavior of defaulted default constructor → invoke the default constructor of its base class and not-static class members

Copy Constructor

- 定義：有一個自己型別的參考(reference)作為參數的建構子

```
class Position{
public:
    Position(const Position&) = default;
};
```

- 只要使用者沒定義，且類別所有成員都具備複製建構子，就會產生預設的複製建構子
- 同樣可以明確指定使用預設與否
- 呼叫方式：

```
Position a;     //default constructor
Position b(a); //copy constructor
```

Copy Constructor

- 使用傳值呼叫的函數時，會對該參數執行一次複製建構子

```
void f(Position p){
    // p is copied by copy constructor
    std::cout << p.x << ", " << p.y << "\n"
}
void main(){
    Position a(0, 0);
    f(a); // Position p(a);
}
```

- `Position a = b; // copy constructor as well`

Explicit Constructor Call Example

- Such a call returns "anonymous object"
  - Which can then be assigned
  - **In Action**:

```
Position heroPos(7, 4);

heroPos = Position(5, 5);
```

Destructors(解構子)

- 定義

```
class Position{
public:
    ~Position() {}
};
```

- 當物件被刪除時執行
  - delete
  - {} 結尾處，刪除區域變數時
- 未明確定義會產生預設解構子
  - 同樣可以明確指定使用預設與否
- 若類別成員有指標需要管理記憶體，必須自行定義解構子釋放之

Constructor + Destructors

```
class Vector{
public:
    Vector(){}
    Vector(const Vector&) = default;
    ~Vector(){}
private:
    float _x=0.0f;
    float _y=0.0f;
};
```

Class Type Member Variables

- Class member variables can be any type
  - Including objects of other classes!
  - Type of class relationship
    - Powerful OOP principle
- Need special notation for constructors
  - So they can call "back" to member object's constructor

Class Member Variable Example: Display 7.3 A Class Member Variable (1 of 5)

Display 7.3 A Class Member Variable

```
1   #include <iostream>
2   #include<cstdlib>
3   using namespace std;

4   class DayOfYear
5   {
6   public:
7       DayOfYear(int monthValue, int dayValue);
8       DayOfYear(int monthValue);
9       DayOfYear( );
10      void input( );
11      void output( );
12      int getMonthNumber( );
13      int getDay( );
14  private:
15      int month;
16      int day;
17      void testDate( );
18  };
```

The class `DayOfYear` *is the same as in Display 7.1, but we have repeated all the details you need for this discussion.*

Class Member Variable Example: Display 7.3 A Class Member Variable (2 of 5)

```
19  class Holiday
20  {
21  public:
22      Holiday( );//Initializes to January 1 with no parking enforcement
23      Holiday(int month, int day, bool theEnforcement);
24      void output( );
25  private:
26      DayOfYear date;
27      bool parkingEnforcement;//true if enforced
28  };

29  int main( )
30  {
31      Holiday h(2, 14, true);
32      cout << "Testing the class Holiday.\n";
33      h.output( );

34      return 0;
35  }
36
37  Holiday::Holiday( ) : date(1, 1), parkingEnforcement(false)
38  {/*Intentionally empty*/}

39  Holiday::Holiday(int month, int day, bool theEnforcement)
40                      : date(month, day), parkingEnforcement(theEnforcement)
41  {/*Intentionally empty*/}
```

member variable of a class type

Invocations of constructors from the class `DayOfYear`.

(continued)

Class Member Variable Example: Display 7.3 A Class Member Variable (3 of 5)

Display 7.3 A Class Member Variable

```
42  void Holiday::output( )
43  {
44      date.output( );
45      cout << endl;
46      if (parkingEnforcement)
47          cout << "Parking laws will be enforced.\n";
48      else
49          cout << "Parking laws will not be enforced.\n";
50  }

51  DayOfYear::DayOfYear(int monthValue, int dayValue)
52                              : month(monthValue), day(dayValue)
53  {
54      testDate( );
55  }
```

Parameter Passing Methods

- Efficiency of parameter passing
  - Call-by-value
    - Requires copy be made → Overhead
  - Call-by-reference
    - Placeholder for actual argument
    - Most efficient method
  - Negligible difference for simple types
  - For class types → clear advantage
- Call-by-reference desirable
  - Especially for "large" data, like class types

The `const` Parameter Modifier

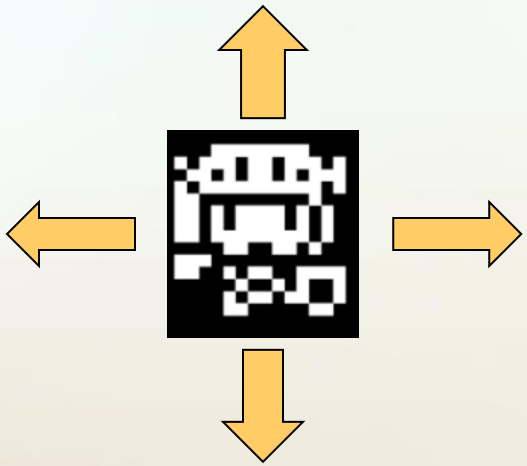
- Large data types (typically classes)
  - Desirable to use pass-by-reference
  - Even if function will not make modifications
- Protect argument
  - Use constant parameter
    - Also called constant call-by-reference parameter
  - Place keyword `const` before type
  - Makes parameter "read-only"
  - Attempt to modify parameter results in compiler error

Use of `const`

- All-or-nothing
- If no need for function modifications
  - Protect parameter with `const`
  - Protect ALL such parameters
- This includes class member function parameters

Example: Direction

- 遊戲中的方向使用(x, y)兩個數值表示，其中又以少部分方向在遊戲中會經常性使用到。
  - 上、下、左、右。
  - 利用 const 將常用到的類別狀態宣告，增加程式可讀性



玩家在移動時可選擇的方向是固定的

const Example :Direction

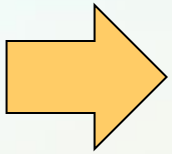
```
1   #pragma once
2   struct Position          //position struct
3   {
4       int x;
5       int y;
6   };
7   int Range(Position a, Position b);
8   bool Equal(Position A, Position B);
9   Position Add(Position A, Position B);
10  Position Reverse(Position A);
11  ////////////
12  //主要區域
13  //const 變數實作
14  //定義常用到的方向，增加程式可讀性
15  ////////////
16  const Position UP{ 0,-1 };
17  const Position DOWN{ 0,1 };
18  const Position RIGHT{ 1,0 };
19  const Position LEFT{ -1,0 };
20  const Position ZERO{ 0,0 };
21  const Position ONE{ 1,1 };
```

位置結構與常用涵式

定義常用到的方向，在程式需要時可直接呼叫，增加可讀性

const Example :Direction

```
Position moveDir = Position{ 0,0 };
if (keyState[validInput::w]) {
    moveDir = Position{ 0,-1 };
    hasInput = true;
}
else if (keyState[validInput::s]) {
    moveDir = Position{ 0,1 };
    hasInput = true;
}
else if (keyState[validInput::a]) {
    moveDir = Position{ -1,0 };
    hasInput = true;
}
else if (keyState[validInput::d]) {
    moveDir = Position{ 1,0 };
    hasInput = true;
}
```



```
Position moveDir = Position{ 0,0 };
if (keyState[validInput::w]) {
    moveDir = UP;
    hasInput = true;
}
else if (keyState[validInput::s]) {
    moveDir = DOWN;
    hasInput = true;
}
else if (keyState[validInput::a]) {
    moveDir = LEFT;
    hasInput = true;
}
else if (keyState[validInput::d]) {
    moveDir = RIGHT;
    hasInput = true;
}
```

利用建構子產生需要的方向

使用const定義好的參數，程式可讀性較佳

Constant Member Function

- The member function can be marked as constant

```
class Position{
public:
    int getX() const { return _x;}
    int getY() const;
    void test() const;
    void test2();
private:
    int _x, _y;
};
int Position::getY() const { return _y; }
```

- For constant member function, no member data can be modified

```
void Position::test() const { _x = 0; } // illegal!
void Position::test() const { test2(); } // illegal!
```

`inline` Functions

- For **non-member** functions:
  - Use keyword ***`inline`*** in function declaration and function heading
- For class **member** functions:
  - Place implementation (code) for function **IN** class definition
    - automatically inline
  - Use inline to define separately, in different file
- Use for very short functions only
- Code actually inserted in place of call
  - Eliminates overhead
  - More efficient, but only when short!
  - If too long → actually less efficient! → compiler may ignore the specifier

Which One Are `inline` Functions

```
1)    class Account
2)    {
3)    public:
4)        Account(double initial_balance) { balance = initial_balance; }
5)        double GetBalance();
6)        double Deposit( double Amount );
7)        double Withdraw( double Amount );
8)    private:
9)        double balance;
10)   };

1)    inline double Account::GetBalance() {    return balance;  }

1)    inline double Account::Deposit( double Amount )
2)    {
3)        return ( balance += Amount );
4)    }

1)    inline double Account::Withdraw( double Amount )
2)    {
3)        return ( balance -= Amount );
4)    }
5)    int main() { … }
```

Inline member function examples

- A function defined in the body of a class declaration is an inline function.
- Example
  - the Account constructor is an inline function.
  - The member functions GetBalance, Deposit, and Withdraw are not specified as inline but can be implemented as inline functions.

`inline` functions vs. macros

- `inline` functions are similar to macros
  - The function code is expanded at the point of the call at compile time
    - `inline` functions are parsed by the compiler
  - macros are expanded by the preprocessor
- Important differences:
  1. `inline` functions follow all the protocols of type safety enforced on normal functions.
     - which one can deliver private class data to outside?
  2. Expressions passed as arguments to inline functions are evaluated once. In some cases, expressions passed as arguments to macros can be evaluated more than once.

Example

```
1)    // inline_functions_macro.c
2)    #include <stdio.h>
3)    #include <ctype.h>
4)    #define toupper(a) ((a) >= 'a' && ((a) <= 'z') ? ((a)-('a'-'A')):(a))
5)    int main() {
6)       char ch;
7)       printf("Enter a character: ");
8)       ch = toupper( getc(stdin) );
9)       printf( "%c", ch );
10)   }
11)   //  Sample Input:  xyz
12)   // Sample Output:  Z
```

getc is executed to determine whether the character is

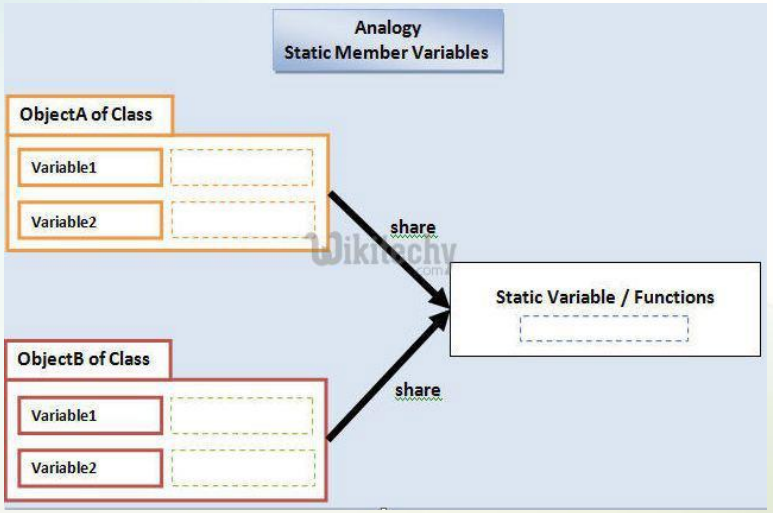
1. >= "a," and
2. <= "z.“
3. converted to uppercase.

```
1)    // inline_functions_inline.cpp
2)    #include <stdio.h>
3)    #include <ctype.h>
4)    inline char toupper( char a ) {
5)       return ((a >= 'a' && a <= 'z') ? a-('a'-'A') : a );
6)    }
7)    int main() {
8)       printf("Enter a character: ");
9)       char ch = toupper( getc(stdin) );
10)      printf( "%c", ch );
11)   }
12)   // Sample Input: a
13)   // Sample Output: A
```

getc is executed once!

Static Members and Functions

- Static member variables
  - All objects of class "share" one copy
  - One object changes it → all see change
- Useful for "tracking"
  - How often a member function is called
  - How many objects exist at given time
- Place keyword *`static`* before type
  - e.g., `static int staticValue = 0;`



- Member functions can be `static`
  - If no access to the member data needed, i.e., cannot access member data
  - And still "must" be the member of the class
  - Make it a `static` function
- Can then be called outside class
  - From non-class objects: `Server::getTurn();`
  - Via class objects: `myObject.getTurn();`
- Can only use static data, functions!

Static Members Example: Display 7.6 Static Members (1 of 4)

Display 7.6 Static Members

```
1   #include <iostream>
2   using namespace std;

3   class Server
4   {
5   public:
6       Server(char letterName);
7       static int getTurn( );
8       void serveOne( );
9       static bool stillOpen( );
10  private:
11      static int turn;
12      static int lastServed;
13      static bool nowOpen;
14      char name;
15  };

16  int Server:: turn = 0;
17  int Server:: lastServed = 0;
18  bool Server::nowOpen = true;
```

Static Members Example: Display 7.6 Static Members (2 of 4)

```
19  int main( )
20  {
21      Server s1('A'), s2('B');
22      int number, count;
23      do
24      {
25          cout << "How many in your group? ";
26          cin >> number;
27          cout << "Your turns are: ";
28          for (count = 0; count < number; count++)
29              cout << Server::getTurn( ) << ' ';
30          cout << endl;
31          s1.serveOne( );
32          s2.serveOne( );
33      } while (Server::stillOpen( ));

34      cout << "Now closing service.\n";

35      return 0;
36  }
37
38
```

Static Members Example: Display 7.6 Static Members (3 of 4)

Display 7.6 Static Members

```
39  Server::Server(char letterName) : name(letterName)
40  {/*Intentionally empty*/}

41  int Server::getTurn( )
42  {
43      turn++;
44      return turn;
45  }
46  bool Server::stillOpen( )
47  {
48      return nowOpen;
49  }

50  void Server::serveOne( )
51  {
52      if (nowOpen && lastServed < turn)
53      {
54          lastServed++;
55          cout << "Server " << name
56              << " now serving " << lastServed << endl;
57       }
```

Since `getTurn` *is static, only static members can be referenced in here.*

Static Members Example: Display 7.6 Static Members (4 of 4)

```
58        if (lastServed >= turn) //Everyone served
59            nowOpen = false;
60    }
```

SAMPLE DIALOGUE

How many in your group? **3**
Your turns are: 1 2 3
Server A now serving 1
Server B now serving 2
How many in your group? **2**
Your turns are: 4 5
Server A now serving 3
Server B now serving 4
How many in your group? **0**
Your turns are:
Server A now serving 5
Now closing service.

Static

- 寫在 class 定義外面的 static 定義：僅屬於該編譯單元的符號
- 編譯單元
  - 一個 .cpp 檔編譯成一個 .o 為一個編譯單元
  - 最終連結器會將所有 .o 以及 .a/.lib 連結成執行檔

Example: 陷阱

- 迷宮中會擺設機關陷阱，妨礙玩家。
  - 當陷阱被觸發時會發出巨響，使的玩家被所有生物發現。
  - 生物間利用靜態變數判斷事件是否被觸發。
  - 機關呼叫生物的靜態函式，告知生物事件觸發。

當陷阱觸發時，會令所有生物發現玩家

生物間利用靜態變數判斷陷阱觸發

Example: 陷阱

```
10  class Creature {//creature class
11  private:
12      int health = 1;      //血量，小於零時代表死亡
13      Position pos;        //位置，表示在版面中的相對位置
14      string icon = "C"; //符號，版面中代表的符號
15      string halfIcon = "C";  //半形符號，處於特殊狀態用的符號。
16      string fullIcon = "C"; //全形符號，處於一般狀態用的符號。
17      int range = 1;           //攻擊範圍，當玩家與其的距離小於該數值
18      bool isDead = false;     //死亡狀態，血量小於零時為true
19      ///////////
20      //主要區域
21      //static 變數
22      //用來判斷事件是否被觸發
23      ///////////
24      static bool autoFindMode; //自動找到主角
```

靜態變數:用以判斷機關觸發

Example: 陷阱

```
25  public:
26      Creature();
27      Creature(int x, int y);
28      Creature(int x, int y, string fullIcon, string halfIcon);
29      bool canSee(Position pos, int& dir_x, int &dir_y);
30      void attack(Hero *h);
31      void move(int x, int y) {
32          this->pos.x += x;
33          this->pos.y += y;
34      }
35      void update(Hero *h);
36      Position getPosition();
37      void getHurt(int number);
38      string getIcon();
39      ///////////
40      //主要區域
41      //static function
42      //機關被觸發時可呼叫此涵式改變靜態變數
43      ///////////
44      static void autoFindHero();
45      static void disautoFindHero();
46  };
```

機關觸發時會呼叫生物的靜態涵式

```
1   #include "Trigger.h"
2   void Trigger::triggered() {
3       cout << "Hero has triggered the Trigger!" << endl
4       activate = true;
5       icon = "  ";
6       Creature::autoFindHero();
7   }
```

```
void Creature::autoFindHero() {
    autoFindMode = true;
}

void Creature::disautoFindHero() {
    autoFindMode = false;
}
```

Example: 陷阱

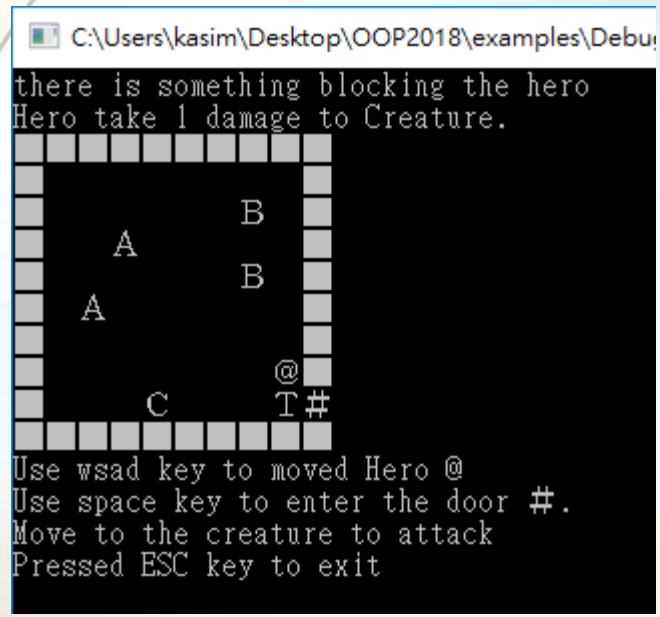
```cpp
#pragma once
#include "Creature.h"
#include<iostream>

using namespace std;
bool Creature::autoFindMode = false;
```

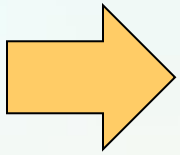
在.cpp檔使用static變數前須在檔案上方宣告變數

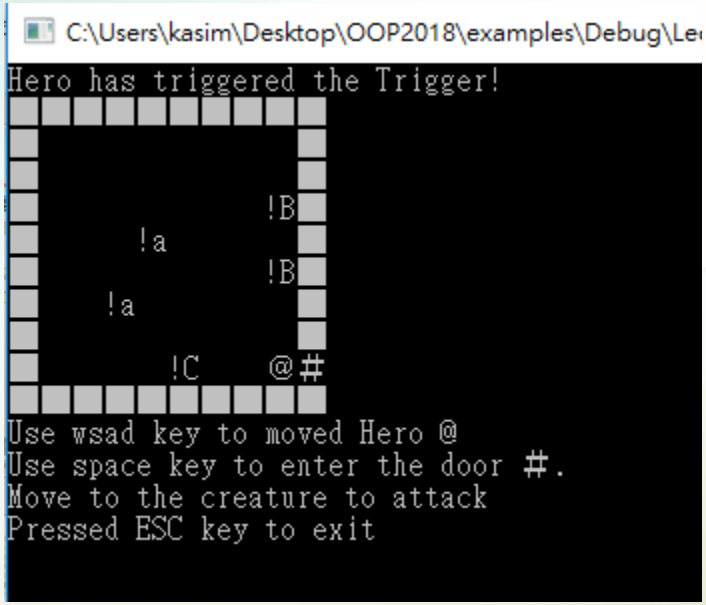
```cpp
//////////
//主要區域
//static 變數
//在偵測主角時若機關已被觸發則直接鎖定主角
//////////
bool Creature::canSee(Position pos, int& dir_x, int &dir_y) {
    //the dir_x and dir_y value are call-by-refernce
    dir_x = clip((pos.x - this->pos.x), -1, 1); //clip the valu
    dir_y = clip((pos.y - this->pos.y), -1, 1);
    if (autoFindMode)
        return true;
    int count = 0;
    do {
        if (this->pos.x + dir_x * count == pos.x &&
            this->pos.y + dir_y * count == pos.y) {
            return true;    //spot the target position
        }
        count++;
    } while (count < 4); //check the range in 4 units
    return false;
}
```

當在偵測主角時若機關已被觸發則直接鎖定主角

Example: 陷阱







陷阱觸發前生物不會追逐在視線外的玩家

陷阱觸發後生物鎖定並追逐玩家

Summary 1

- Constructors: automatic initialization of class data
  - Called when objects are declared
  - Constructor has same name as class
- Default constructor has no parameters
  - Should always be defined
- Class member variables
  - Can be objects of other classes
    - Require initialization-section

Summary 2

- Constant call-by-reference parameters
  - More efficient than call-by-value
- Can *inline* very short function definitions
  - Can improve efficiency
- Static member variables
  - Shared by all objects of a class
- Vector classes
  - Like: "arrays that grow and shrink"