Chapter 3

Function Basics

Learning Objectives

- Predefined Functions (standard library)
  - Those that return a value and those that don't
- Programmer-defined Functions
  - Defining, Declaring, Calling
  - Recursive Functions
- Scope Rules
  - Local variables
  - Global constants and global variables
  - Blocks, nested scopes

Introduction to Functions

- Building Blocks of Programs
- Other terminology in other languages:
  - Procedures, subprograms, methods
  - In C++: functions
- **I-P-O**
  - Input – Process – Output
  - Basic subparts to any program
  - Use functions for these "pieces"

Predefined Functions

- Libraries full of functions for our use!
- Two types:
  - Those that return a value
  - Those that do not (void)
- Must `"#include"` appropriate library
  - e.g.,
    - `<cmath>, <cstdlib>` (Original "C" libraries)
    - `<iostream>` (for cout, cin)

Using Predefined Functions

- Math functions very plentiful
  - Found in library `<cmath>`
  - Most return a value (the "answer")
- Example: **`theRoot`** `=` **`sqrt`**`(9.0);`
  - Components:

    `sqrt` = name of library function

    `theRoot` = variable used to assign "answer" to

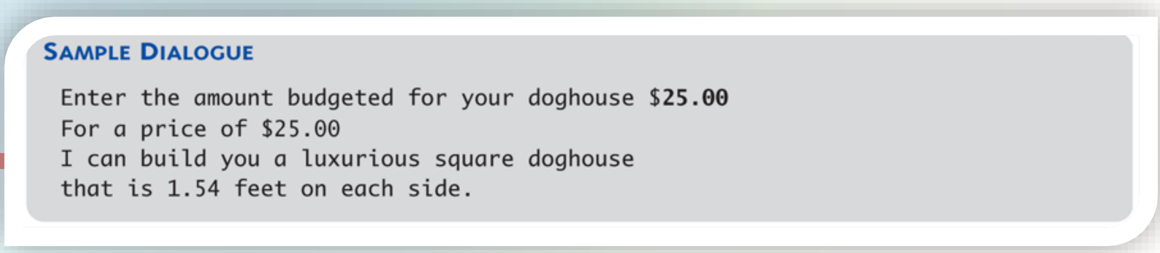
    `9.0` = argument or "starting input" for function
  - In **I-P-O**:
    - I = 9.0
    - P = "compute the square root"
    - O = 3, which is returned & assigned to theRoot

The Function Call

- Back to this assignment:

```
theRoot = sqrt(9.0);
```

  - The expression "`sqrt(9.0)`" is known as a function *call*, or function *invocation*
  - The argument in a function call (9.0) can be a literal, a variable, or an expression
  - The call itself can be part of an expression:
    - `bonus = sqrt(sales)/10;`
    - A function call is allowed wherever it's legal to use an expression of the function's return type



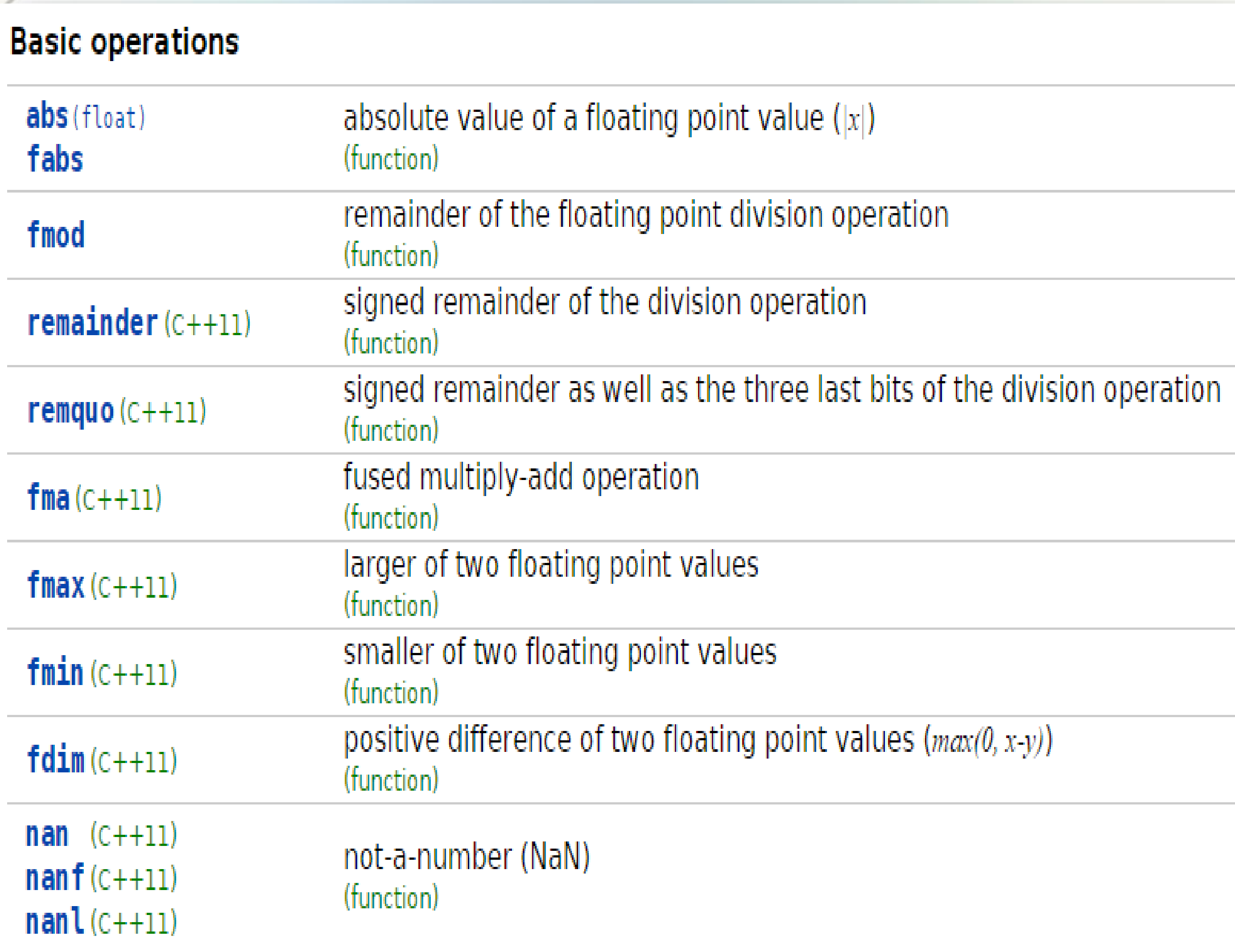
SAMPLE DIALOGUE

```
Enter the amount budgeted for your doghouse $25.00
For a price of $25.00
I can build you a luxurious square doghouse
that is 1.54 feet on each side.
```

```
#include <iostream>
#include <cmath>
using namespace std;
int main( )
{
    const double COST_PER_SQ_FT = 10.50;
    double budget, area, lengthSide;

    cout << "Enter the amount budgeted for your dog house $";
    cin >> budget;
    area = budget/COST_PER_SQ_FT;
    lengthSide = sqrt(area);

    cout.setf(ios::fixed);
    cout.setf(ios::showpoint);
    cout.precision(2);
    cout << "For a price of $" << budget << endl
        << "I can build you a luxurious square dog house\n"
        << "that is " << lengthSide
        << " feet on each side.\n";

    return 0;
}
```

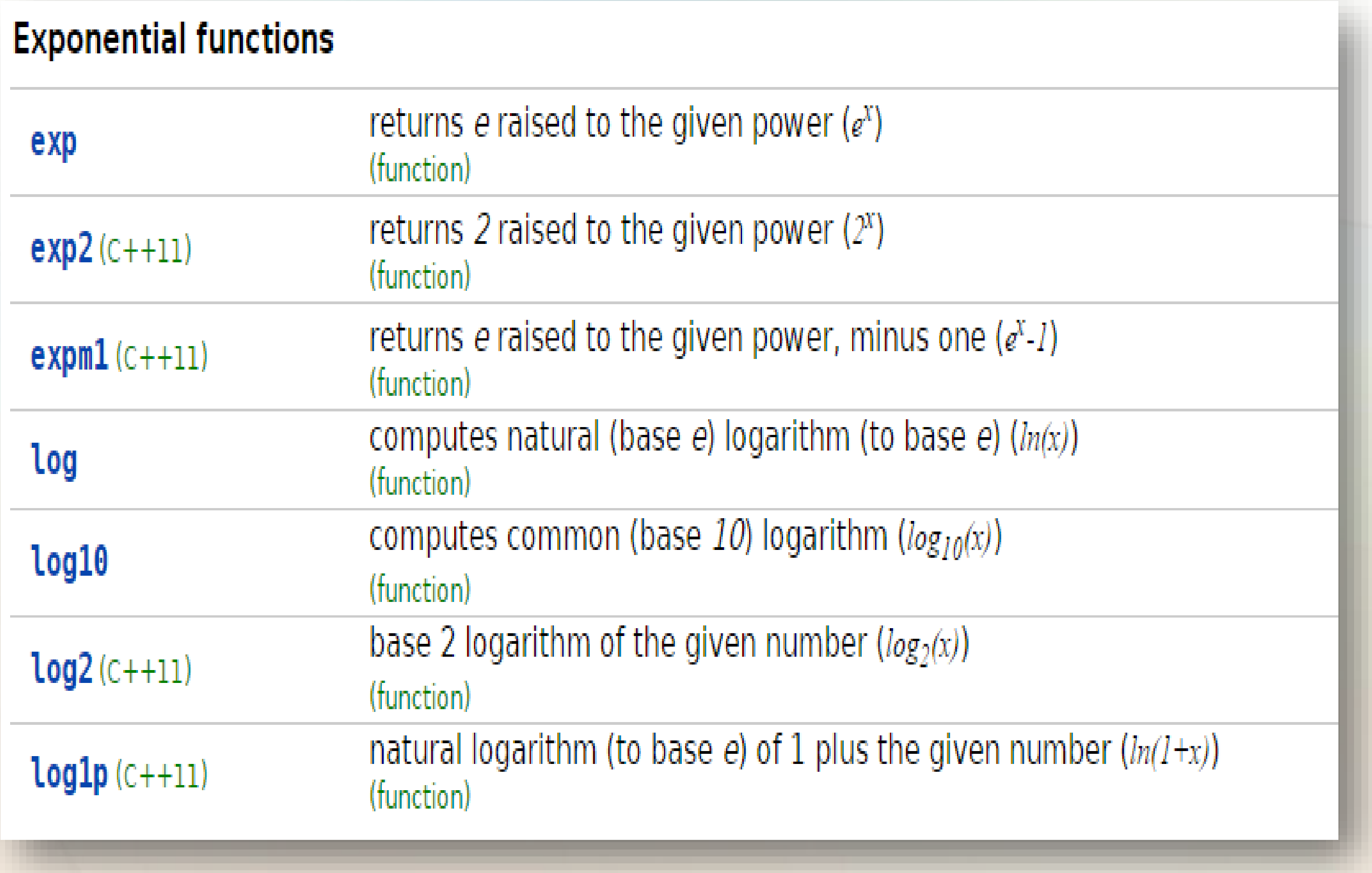
Notes on Display 3.1

- Cmath functions
  - http://en.cppreference.com/w/cpp/header/cmath



Basic operations

abs (float) **fabs**	absolute value of a floating point value ($\|x\|$) (function)
fmod	remainder of the floating point division operation (function)
remainder (C++11)	signed remainder of the division operation (function)
remquo (C++11)	signed remainder as well as the three last bits of the division operation (function)
fma (C++11)	fused multiply-add operation (function)
fmax (C++11)	larger of two floating point values (function)
fmin (C++11)	smaller of two floating point values (function)
fdim (C++11)	positive difference of two floating point values ($max(0, x-y)$) (function)
nan (C++11) **nanf** (C++11) **nanl** (C++11)	not-a-number (NaN) (function)

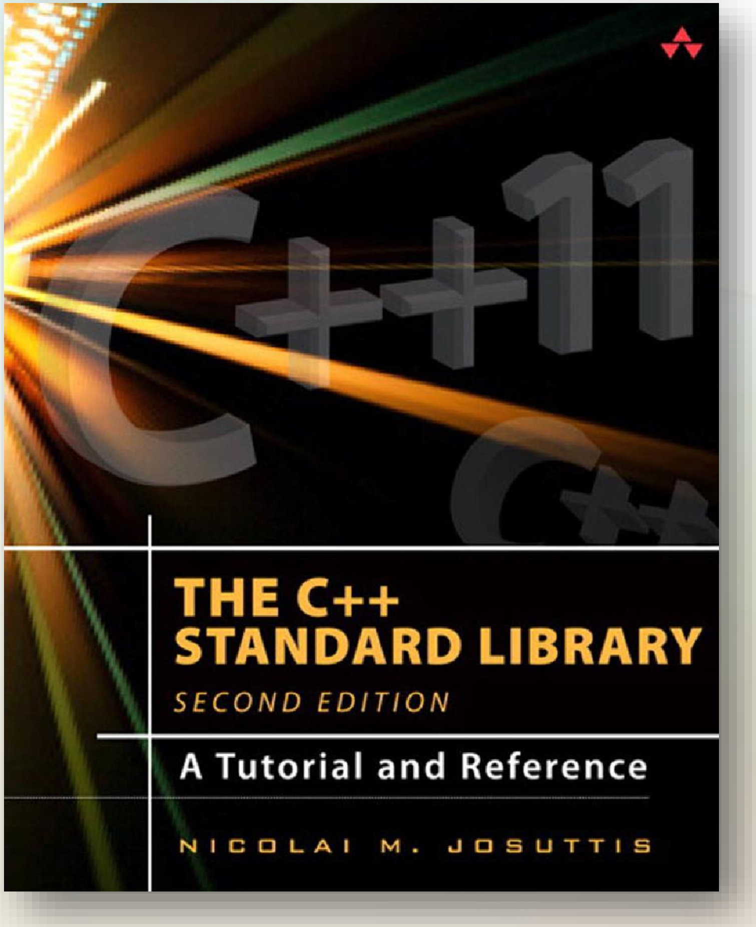


Exponential functions

exp	returns *e* raised to the given power (e^x) (function)
exp2 (C++11)	returns *2* raised to the given power (2^x) (function)
expm1 (C++11)	returns *e* raised to the given power, minus one (e^x-1) (function)
log	computes natural (base *e*) logarithm (to base *e*) ($ln(x)$) (function)
log10	computes common (base *10*) logarithm ($log_{10}(x)$) (function)
log2 (C++11)	base 2 logarithm of the given number ($log_2(x)$) (function)
log1p (C++11)	natural logarithm (to base *e*) of 1 plus the given number ($ln(1+x)$) (function)

C++ Standard Library

- http://en.cppreference.com/w/cpp/header



More Predefined Functions

- `#include <cstdlib>`
  - Library contains functions like:
    - `abs()// Returns absolute value of an int`
    - `labs()      // Returns absolute value of a long int`
    - `*fabs()     // Returns absolute value of a float`
  - `*fabs()` is actually in library `<cmath>`!
    - Can be confusing
    - Remember: libraries were added after C++ was "born," in incremental phases
    - Refer to appendices/manuals for details

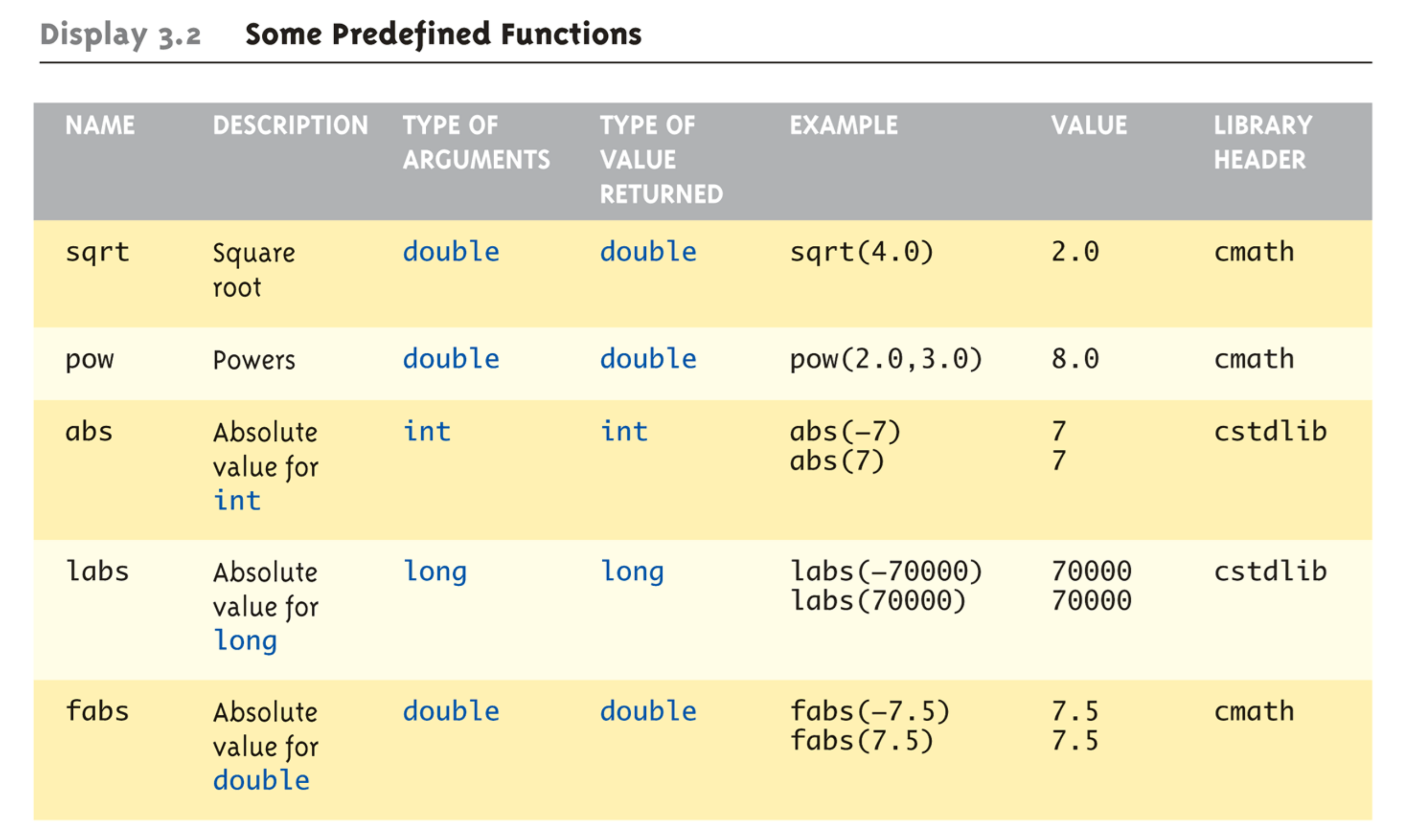
More Math Functions

- `pow(x, y) sqrt(x)`
  - Returns distance to (0, 0) of (x, y)

```
double distance, x = 3.0, y = 2.0;
distance = sqrt (pow(x, 2) + pow(y, 2));
cout << distance;
```

  - Notice this function receives two arguments
  - A function can have any number of arguments, of varying data types

Even More Math Functions: Display 3.2 Some Predefined Functions (1 of 2)



Display 3.2 **Some Predefined Functions**

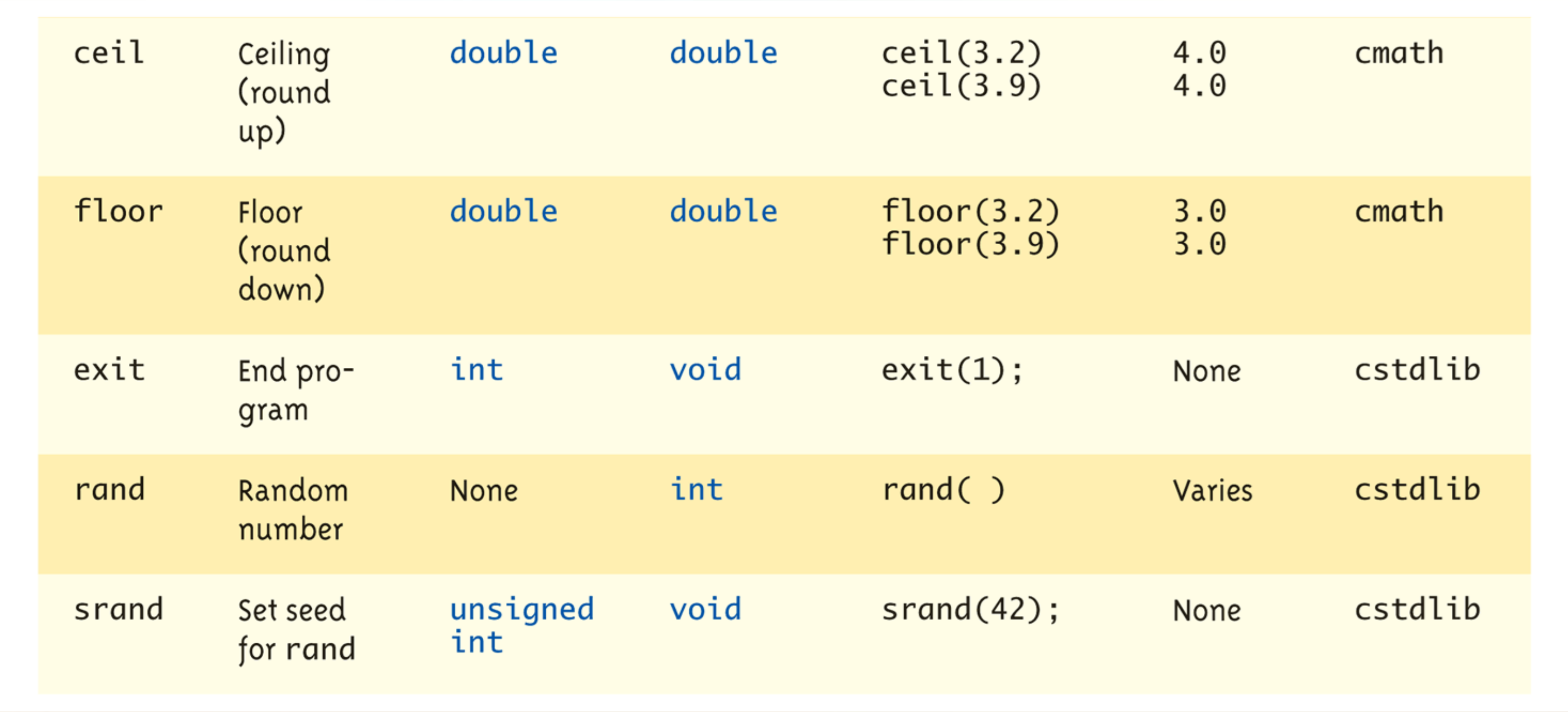
NAME	DESCRIPTION	TYPE OF ARGUMENTS	TYPE OF VALUE RETURNED	EXAMPLE	VALUE	LIBRARY HEADER
sqrt	Square root	double	double	sqrt(4.0)	2.0	cmath
pow	Powers	double	double	pow(2.0,3.0)	8.0	cmath

abs	Absolute value for int	int	int	abs(-7)
				abs(7)
					7
					7
						cstdlib

labs	Absolute value for long	long	long	labs(-70000)
				labs(70000)
					70000
					70000
						cstdlib

fabs	Absolute value for double	double	double	fabs(-7.5)
				fabs(7.5)
					7.5
					7.5
						cmath

Even More Math Functions: Display 3.2 Some Predefined Functions (2 of 2)



ceil	Ceiling (round up)	double	double	ceil(3.2)
				ceil(3.9)
					4.0
					4.0
						cmath

floor	Floor (round down)	double	double	floor(3.2)
				floor(3.9)
					3.0
					3.0
						cmath

exit	End program	int	void	exit(1);	None	cstdlib
rand	Random number	None	int	rand( )	Varies	cstdlib
srand	Set seed for rand	unsigned int	void	srand(42);	None	cstdlib

Predefined Void Functions

- No returned value
- Performs an action, but sends no "answer"
- When called, it's a statement itself
  - `exit(1);// No return value, so not assigned`
    - This call terminates program
    - void functions can still have arguments
- All aspects same as functions that "return a value"
  - They just don't return a value!

Random Number Generator

- Return "randomly chosen" number
- Used for simulations, games
  - `rand()`
    - Takes no arguments
    - Returns value between **0** & **RAND_MAX**
  - Scaling
    - Squeezes random number into smaller range
      `rand() % 6`
    - Returns random value between 0 & 5
  - Shifting
    `rand() % 6 + 1`
    - Shifts range between 1 & 6 (e.g., die roll)

Random Number Seed

- Pseudorandom numbers
  - Calls to `rand()` produce given “sequence” of random numbers
- Use "seed" to alter sequence
  `srand(seed_value);`
  - void function
  - Receives one argument, the "seed"
  - Can use any seed value, including system time:
    `srand(time(0));`
  - `time()` returns system time as numeric value
  - Library `<time>` contains `time()` functions

Random Examples

- Random double between 0.0 & 1.0:

```
(RAND_MAX - rand())/static_cast<double>(RAND_MAX)
```

  - Type cast used to force double-precision division

- Random int between 1 & 6:

```
rand() % 6 + 1
```

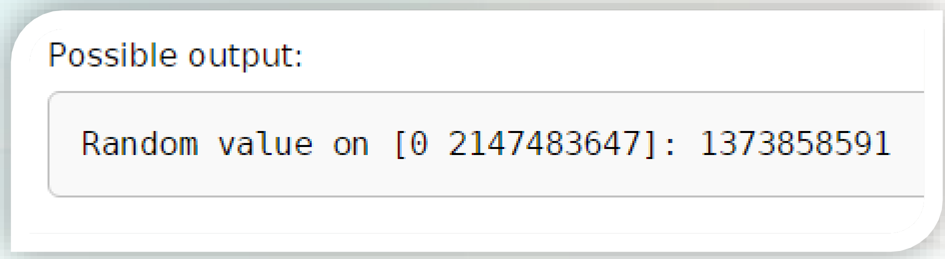
  - "%" is modulus operator (remainder)

- Random int between 10 & 20:

```
rand() % 10 + 10
```

Standard library Examples

```
#include <cstdlib>
#include <iostream>
#include <ctime>

int main()
{
    std::srand(std::time(0)); // use current time
                              // as seed for random
                              // generator
    int random_variable = std::rand();
    std::cout << "Random value on [0 " << RAND_MAX << "]: "
              << random_variable << '\n';
    return 0;
}
```



Possible output:

```
Random value on [0 2147483647]: 1373858591
```

想一想

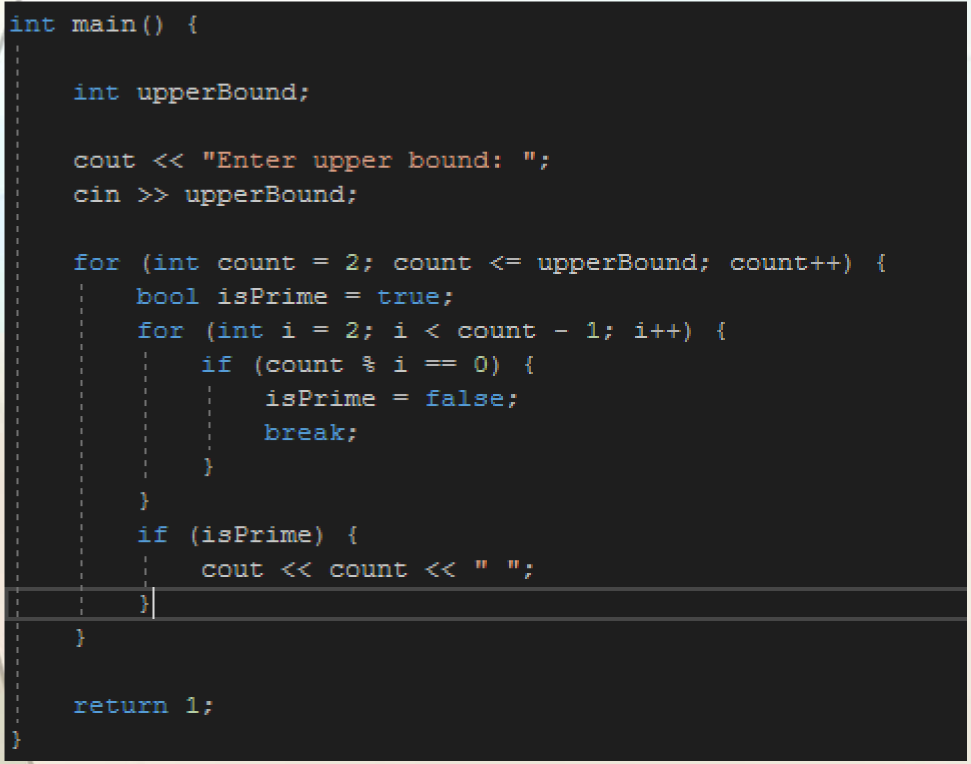
- 遊戲中，反覆出現的功能有那些?
- 要如何處理他們呢?
- 如何減少開發時間及除錯時間呢?

想一想

- 遊戲中，反覆出現的功能有那些?
  - 取得使用者的輸入，移動。觸發後的反應……
- 要如何處理他們呢？
  - 使用相同的Function做相同的事情重複call
- 如何減少開發時間及除錯時間呢?
  - 若把功能做成Function可以減少相同Code的出現，就不會重複打類似的東西並且若出錯只需要針對Function做修正

想一想

- How to improve this?

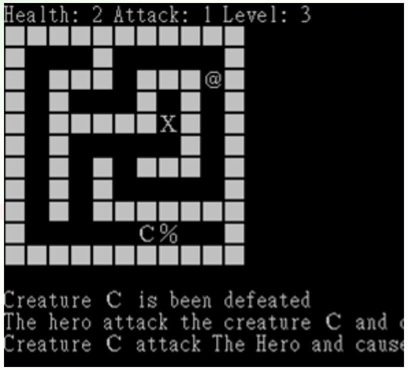


```
int main() {

    int upperBound;

    cout << "Enter upper bound: ";
    cin >> upperBound;

    for (int count = 2; count <= upperBound; count++) {
        bool isPrime = true;
        for (int i = 2; i < count - 1; i++) {
            if (count % i == 0) {
                isPrime = false;
                break;
            }
        }
        if (isPrime) {
            cout << count << " ";
        }
    }

    return 1;
}
```

Library與Function

- function定義了常用的演算法，在程式中重複利用，省去程式中冗贅的部分。這些function更可以被包裝成library提供給其他專案，節省開發時間。
- 利用第三方或C++皆提供的library，寫程式不用/不要全部自己刻。
- 遊戲中用到的C++與Visual Studio提供的function
  - `int  rand()`
    - C標準函式庫->stdlib.h
    - 用於產生隨機數值，在建立隨機迷宮或是決定生物行為都很有用。
  - `SHORT GetAsyncKeyState(int key)`
    - User32.lib -> Winuser.h
    - 用於偵測鍵盤被按下，比cin更適用於角色操作。

Library與Function



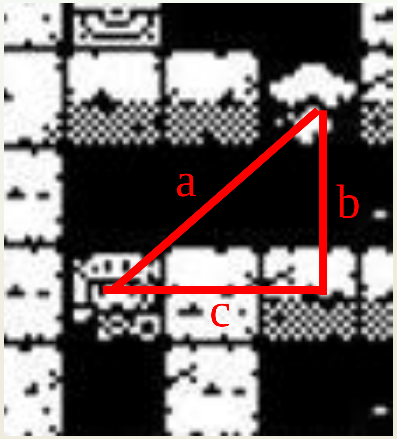
```
Health: 2 Attack: 1 Level: 3
■■■■■■■■■■■
■   ■     ■
■ ■■■ ■■■@■
■ ■   ■ ■ ■
■ ■■■■■X■ ■
■ ■     ■ ■
■ ■ ■ ■■■ ■
■ ■ ■     ■
■ ■ ■■■■■■■
■     C%  ■
■■■■■■■■■■■

Creature C is been defeated
The hero attack the creature C and c
Creature C attack The Hero and cause
```

- 遊戲中定義的function
  - void draw()
    - 在遊戲需要更新畫面時用到，會將輸出視窗刷新並一行一行cout輸出遊戲畫面。
  - string** createCanvas(int width,int height)
    - 函式實作了產生地圖的演算法並輸出二維的陣列資料。該資料在遊戲中的環境判斷與畫面輸出都會用到。
- 在實驗室中常用的方便library
  - OpenGL(開放)，用於繪製2D和3D畫面的圖形資料庫。
    - 提供在GPU上繪製2D和3D圖形的圖形處理函式，在開發圖形化程式時極為有用。
  - OpenCV(學術開放)，跨平台的電腦視覺資料庫。
    - 有非常豐富的圖片處理函式，包括大小轉換、顏色轉換和邊界偵測等各種功能。

Cmath Examples:攻擊英雄

- 遊戲中生物需要計算與主角的距離，若距離在攻擊範圍內則攻擊主角。
- 計算兩點間的距離可利用<Math.h> 擁有的數學函式 sqrt() 與 pow()計算。

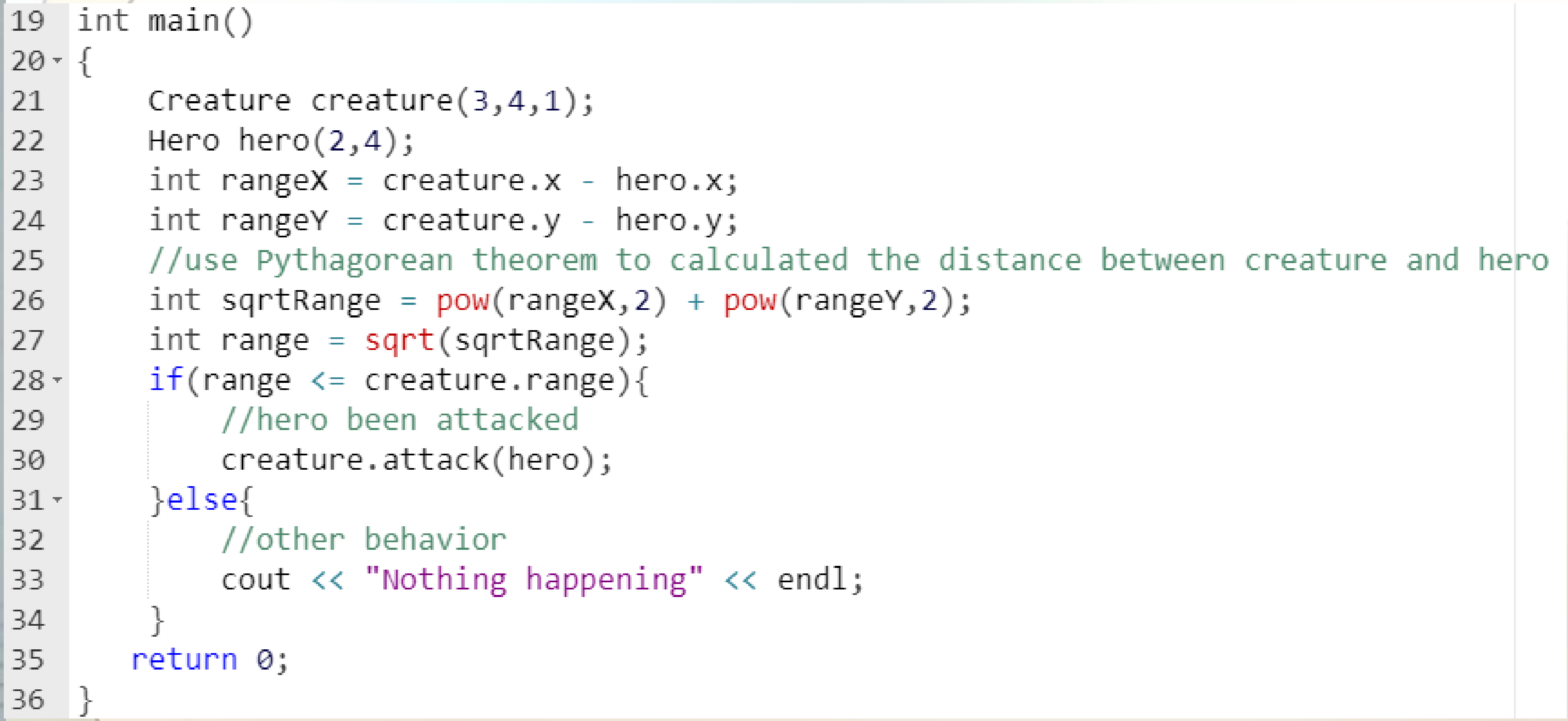
生物在行動時會先判斷主角與自己的距離，若在自己的攻擊範圍內則對主角造成傷害。



生物與主角的距離可用勾股定理計算。

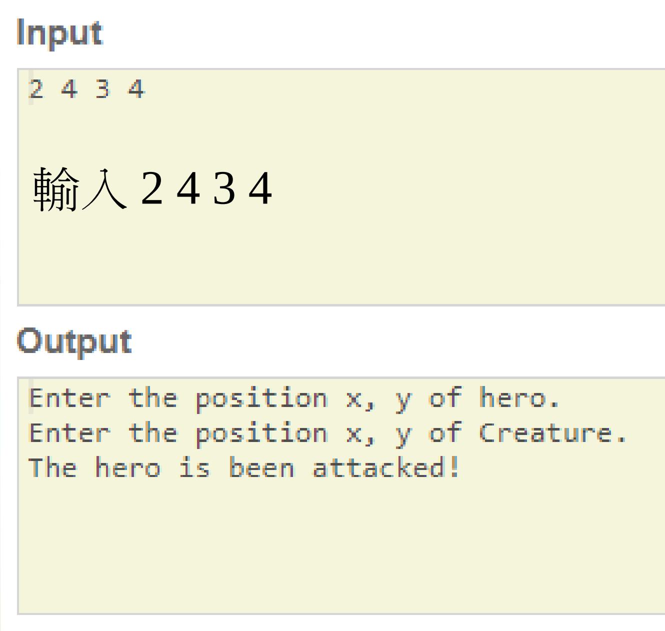
$$a^2 = b^2 + c^2$$

攻擊英雄:Example Code



```
int main()
{
    Creature creature(3,4,1);
    Hero hero(2,4);
    int rangeX = creature.x - hero.x;
    int rangeY = creature.y - hero.y;
    //use Pythagorean theorem to calculated the distance between creature and hero
    int sqrtRange = pow(rangeX,2) + pow(rangeY,2);
    int range = sqrt(sqrtRange);
    if(range <= creature.range){
        //hero been attacked
        creature.attack(hero);
    }else{
        //other behavior
        cout << "Nothing happening" << endl;
    }
   return 0;
}
```

攻擊英雄:Output



Input

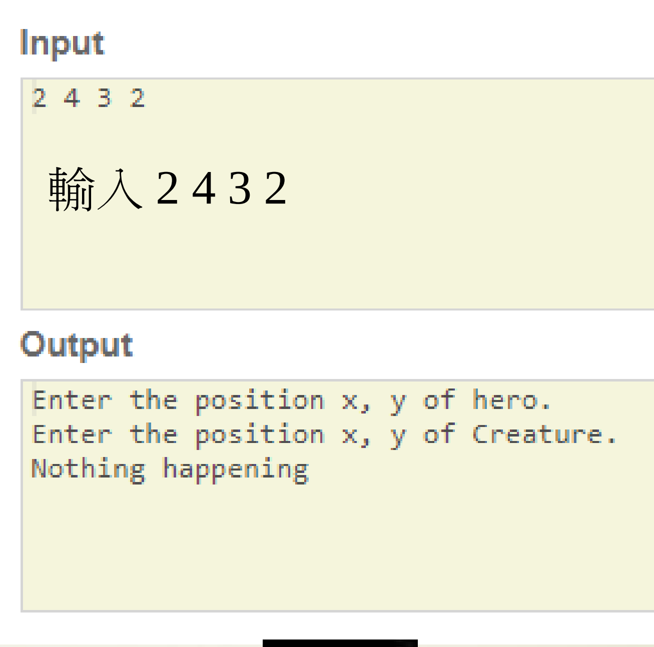
```
2 4 3 4
```

輸入 2 4 3 4

Output

```
Enter the position x, y of hero.
Enter the position x, y of Creature.
The hero is been attacked!
```

生物在主角身旁，並攻擊主角。



Input

```
2 4 3 2
```

輸入 2 4 3 2

Output

```
Enter the position x, y of hero.
Enter the position x, y of Creature.
Nothing happening
```

生物與主角錯開，沒發生攻擊事件。

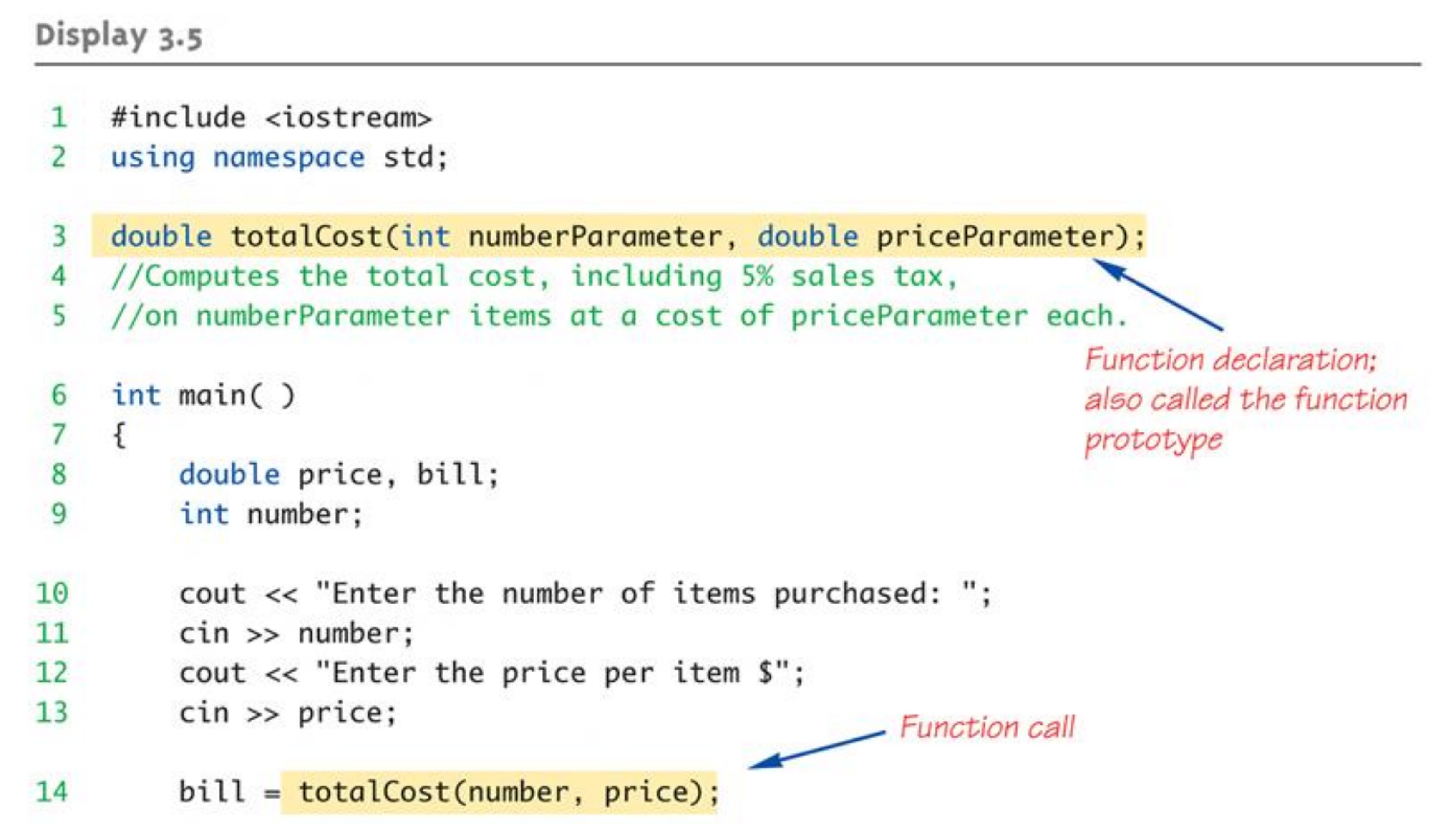
Programmer-Defined Functions

- Write your own functions!
- Building blocks of programs
  - Divide & Conquer
  - Readability
  - Re-use
- Your "definition" can go in either:
  - Same file as `main()`
  - Separate file so others can use it, too

Components of Function Use

- 3 Pieces to using functions:
  - **Function Declaration/prototype**
    - Information for compiler
    - To properly interpret calls
  - **Function Definition**
    - Actual implementation/code for what function does
  - **Function Call**
    - Transfer control to function

Function Example: Display 3.5 A Function to Calculate Total Cost (1 of 2)



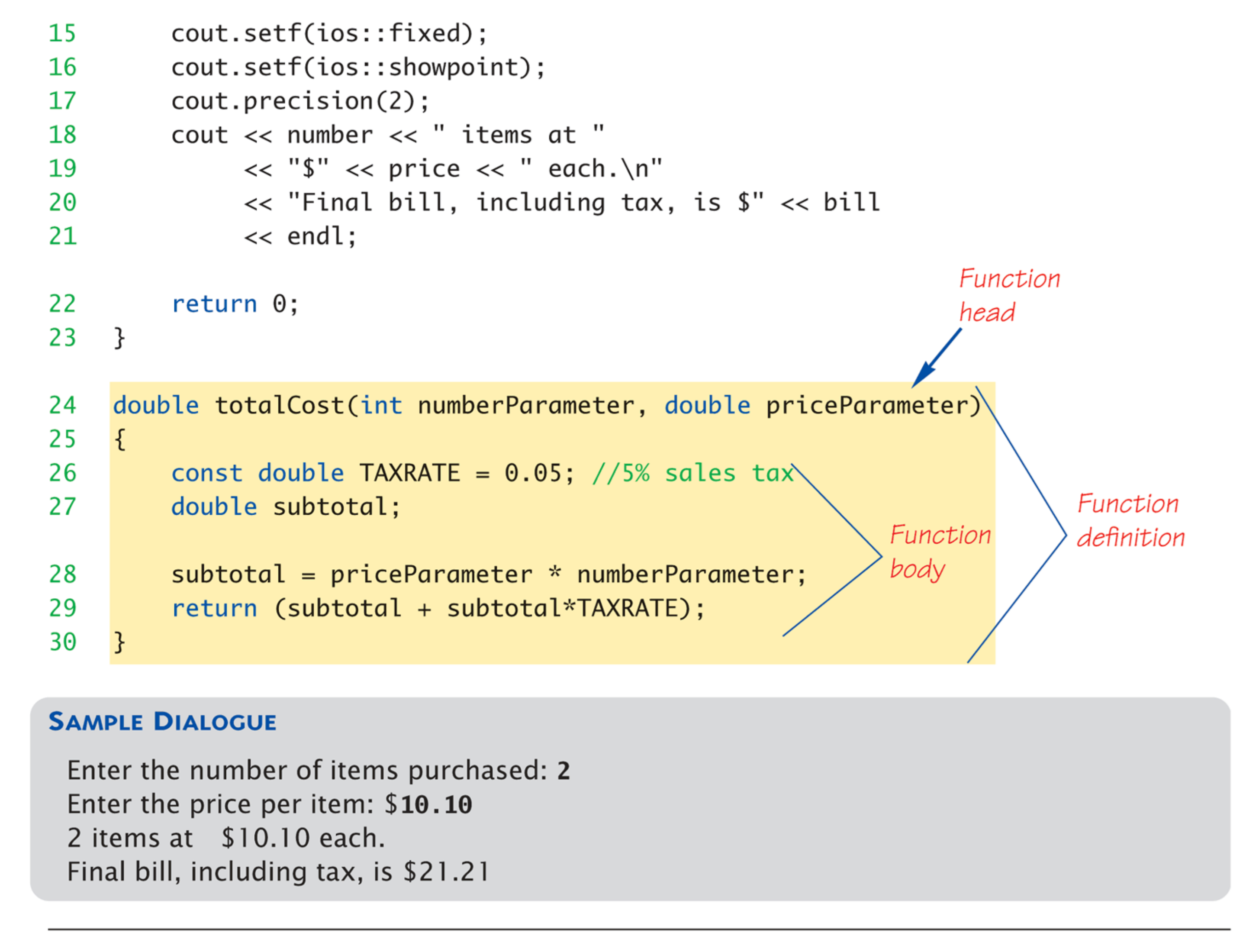
Display 3.5

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

double totalCost(int numberParameter, double priceParameter);
//Computes the total cost, including 5% sales tax,
//on numberParameter items at a cost of priceParameter each.

int main( )
{
    double price, bill;
    int number;

    cout << "Enter the number of items purchased: ";
    cin >> number;
    cout << "Enter the price per item $";
    cin >> price;

    bill = totalCost(number, price);
```

Function declaration; also called the function prototype

Function call

Function Example: Display 3.5 A Function to Calculate Total Cost (1 of 2)



```
    cout.setf(ios::fixed);
    cout.setf(ios::showpoint);
    cout.precision(2);
    cout << number << " items at "
         << "$" << price << " each.\n"
         << "Final bill, including tax, is $" << bill
         << endl;

    return 0;
}

double totalCost(int numberParameter, double priceParameter)
{
    const double TAXRATE = 0.05; //5% sales tax
    double subtotal;

    subtotal = priceParameter * numberParameter;
    return (subtotal + subtotal*TAXRATE);
}
```

Function head

Function body

Function definition

Sample Dialogue

```
Enter the number of items purchased: 2
Enter the price per item: $10.10
2 items at   $10.10 each.
Final bill, including tax, is $21.21
```

Function Declaration

- Also called function prototype
- An "informational" declaration for compiler
- Tells compiler how to interpret calls
  - Syntax:
    `<return_type> FnName(<formal-parameter-list>);`
  - Example:
    `int updateHealth(int HP, double distance);`
  - Placed in the declaration space of `main()`
  - Or above `main()` in global space

Function Definition

- Implementation of function
- Just like implementing function main()
- Example:

```
int updateHealth( int HP, double distance)
{
      const double attack = 5.0;
      double newHealth;
      newHealth = HP – (attack / distance);
      return (int)newHealth;
}
```

- Notice proper indenting

Function Definition Placement

- Placed after function `main()`
  - NOT "inside" function `main()` !
- Functions are “equals”; no function is ever "part" of another
- Formal parameters in definition
  - "Placeholders" for data sent in
    - "Variable name" used to refer to data in definition
- `return` statement
  - Sends data back to caller

Function Call

- Just like calling predefined function
  `health = updateHealth(HP, distance);`
- Recall: `updateHealth` returns int value
  - Assigned to variable named “health"
- Arguments here: HP, distance
  - Recall arguments can be literals, variables, expressions, or combination
  - In function call, arguments often called "actual arguments"
    - Because they contain the "actual data" being sent

Alternative Function Declaration

- Recall: Function declaration is "information“ for compiler
- Compiler only needs to know:
  - Return type
  - Function name
  - Parameter list
- Formal parameter names not needed:

```
double updateHealth(int, double);
```

  - Still "should" put in formal parameter names
    - Improves readability

Parameter vs. Argument

- Terms often used interchangeably
- Formal parameters/arguments
  - In function declaration
  - In function definition’s header
- Actual parameters/arguments
  - In function call
- Technically parameter is "formal" piece while argument is "actual" piece*
  - *Terms not always used this way

Functions Calling Functions

- We’re already doing this!
  - `main()` IS a function!
- Only requirement:
  - Function’s declaration must appear first
- Function’s definition typically elsewhere
  - After `main()`"s definition
  - Or in separate file
- Common for functions to call many other functions
- Function can even call itself → "Recursion"

Boolean Return-Type Functions

- Return-type can be any valid type
  - Given function declaration/prototype:

    ```
    bool isAlive(int HP);
    ```

  - And function’s definition:

    ```
    bool isAlive(int HP)
    {
            return (HP > 0);
    }
    ```

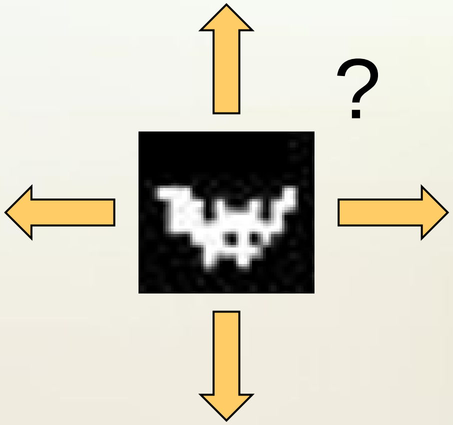
  - Returns "`true`" or "`false`"
  - Function call, from some other function:

    ```
    if (isAlive(HP))
         cout << “Creature is alive\n";
    ```

Example (Contd.)

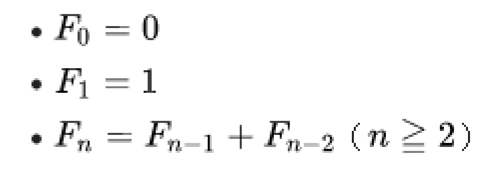
- 隨機移動

```
void randomCreatureMovement(int *creaturePosX, int *creaturePosY)
{
    srand(time(NULL));
    int random = rand();
    if (random % 4 == 0)
        *creaturePosX += 1;
    if (random % 4 == 1)
        *creaturePosX -= 1;
    if (random % 4 == 2)
        *creaturePosY += 1;
    if (random % 4 == 3)
        *creaturePosY -= 1;
}
```



Example (Contd.)

```
#include <iostream>

unsigned fibonacci(unsigned n){
    if (n < 2) return n;
    return fibonacci(n-1) + fibonacci(n-2);
}

int main(){
    unsigned r;
    while(std::cin >> r){
        std::cout << fibonacci(r) << “\n”;
    }
    return 0;
}
```



- $F_0 = 0$
- $F_1 = 1$
- $F_n = F_{n-1} + F_{n-2}$ $(n \geqq 2)$

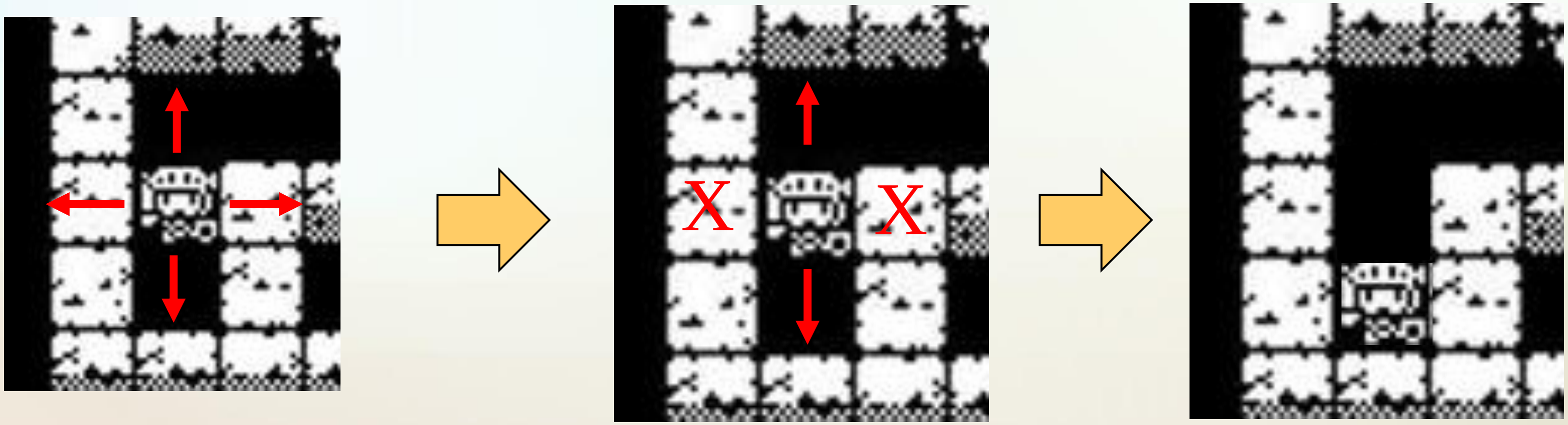
Example (Contd.)

```
f(42) = 267914296
finished computation at Mon Jul 29 08:41:09
2013
elapsed time: 0.670427s
```

```cpp
#include <iostream>
#include <chrono>
#include <ctime>

long fibonacci(unsigned n){
    if (n < 2) return n;
    return fibonacci(n-1) + fibonacci(n-2);
}

int main(){
    std::chrono::time_point<std::chrono::system_clock> start, end;
    start = std::chrono::system_clock::now();
    std::cout << "f(42) = " << fibonacci(42) << '\n';
    end = std::chrono::system_clock::now();

    std::chrono::duration<double> elapsed_seconds = end-start;
    std::time_t end_time = std::chrono::system_clock::to_time_t(end);

    std::cout << "finished computation at " << std::ctime(&end_time)
              << "elapsed time: " << elapsed_seconds.count() << "s\n";
    return 0;
}
```

Function Example: heroMove

- 主角在遊戲中會接收方向移動的指令heroMove(int x, int y)，並回傳bool確認是否成功移動。
- 主角在移動時需判斷移動位置是否為空地，否則停止移動，

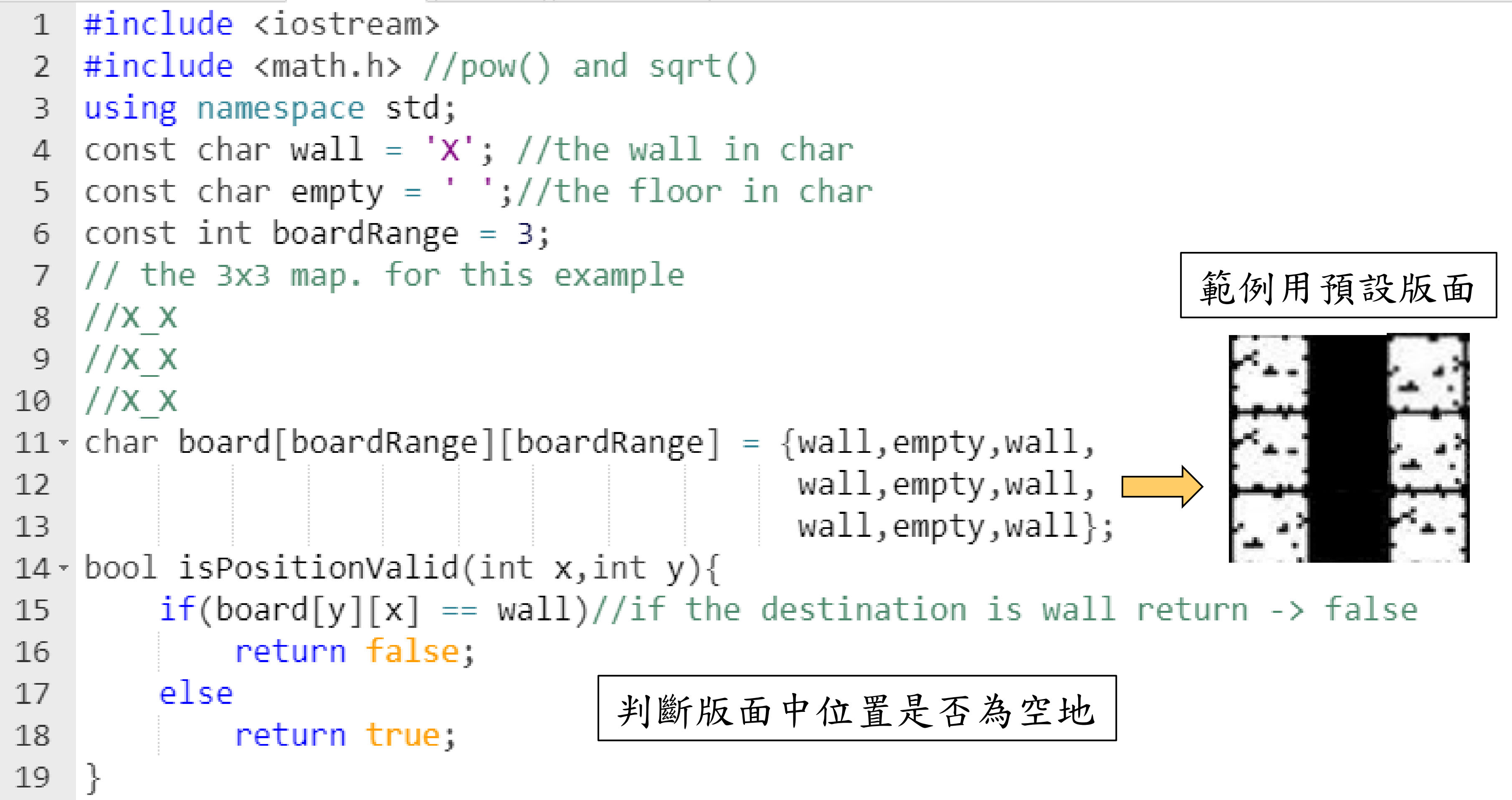


根據輸入判斷移動方向

根據判斷目的地是否為空地，若該方向被擋住則取消操作

移動主角

heroMove:Example Code

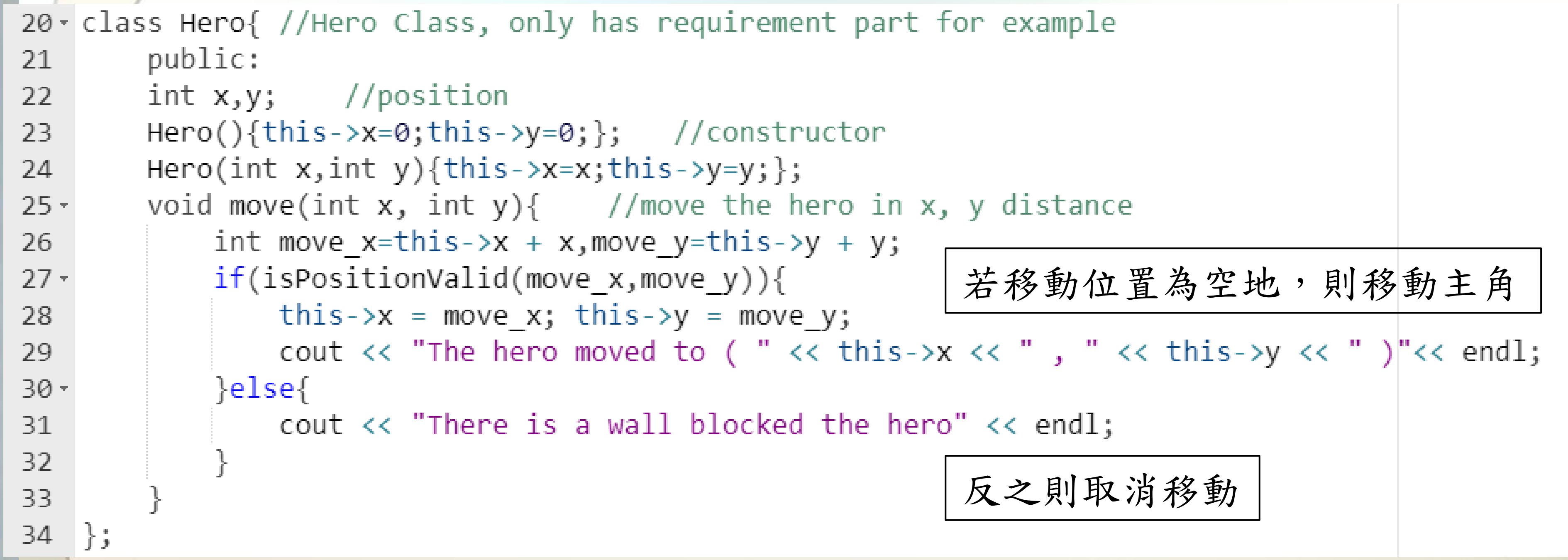


```cpp
#include <iostream>
#include <math.h> //pow() and sqrt()
using namespace std;
const char wall = 'X'; //the wall in char
const char empty = ' ';//the floor in char
const int boardRange = 3;
// the 3x3 map. for this example
//X_X
//X_X
//X_X
char board[boardRange][boardRange] = {wall,empty,wall,
                                      wall,empty,wall,
                                      wall,empty,wall};
bool isPositionValid(int x,int y){
    if(board[y][x] == wall)//if the destination is wall return -> false
        return false;
    else
        return true;
}
```

範例用預設版面

判斷版面中位置是否為空地

heroMove:Example Code

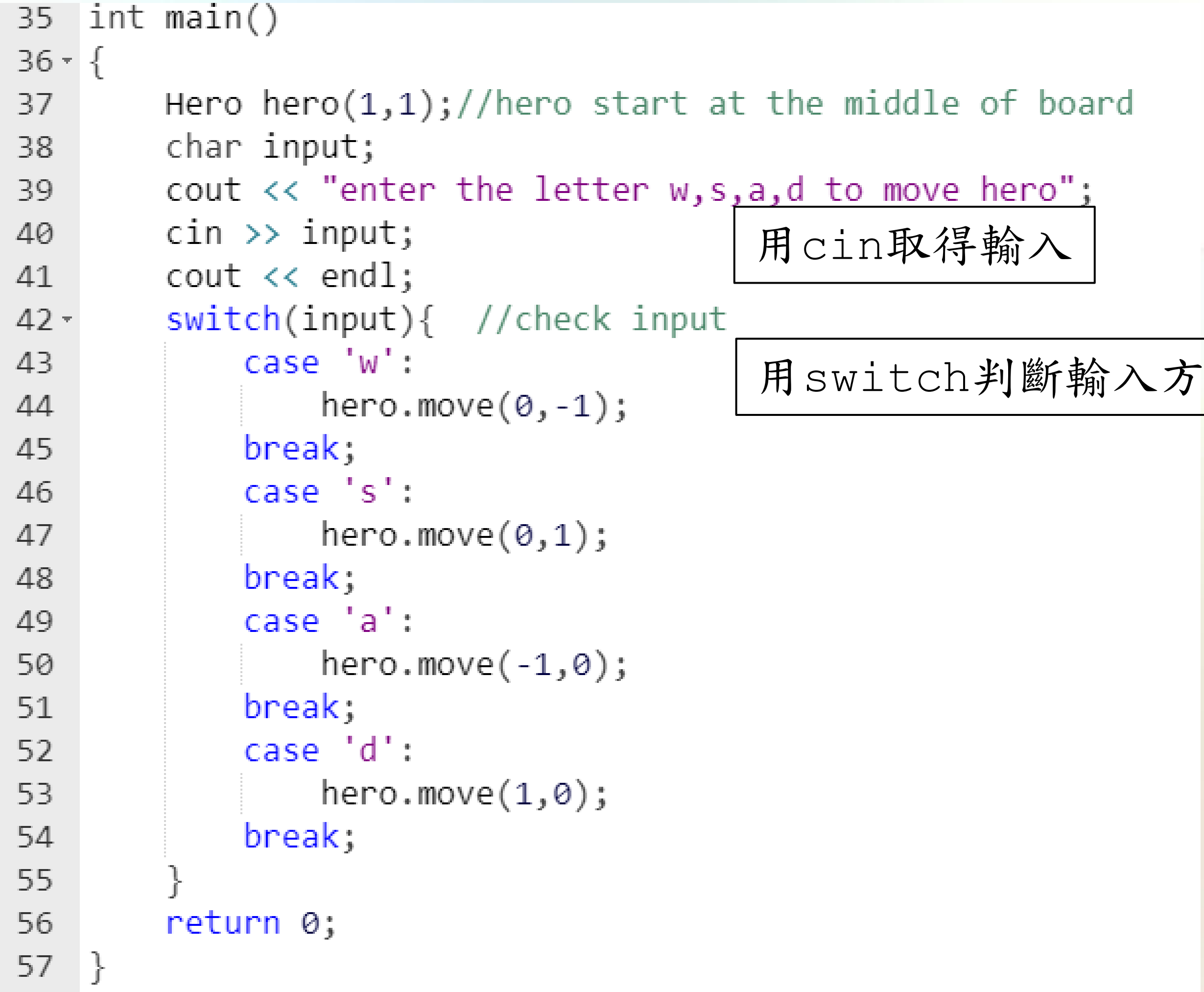


```
class Hero{ //Hero Class, only has requirement part for example
    public:
    int x,y;    //position
    Hero(){this->x=0;this->y=0;};   //constructor
    Hero(int x,int y){this->x=x;this->y=y;};
    void move(int x, int y){    //move the hero in x, y distance
        int move_x=this->x + x,move_y=this->y + y;
        if(isPositionValid(move_x,move_y)){
            this->x = move_x; this->y = move_y;
            cout << "The hero moved to ( " << this->x << " , " << this->y << " )"<< endl;
        }else{
            cout << "There is a wall blocked the hero" << endl;
        }
    }
};
```

若移動位置為空地，則移動主角

反之則取消移動

heroMove:Example Code

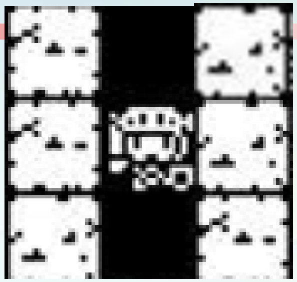


```cpp
int main()
{
    Hero hero(1,1);//hero start at the middle of board
    char input;
    cout << "enter the letter w,s,a,d to move hero";
    cin >> input;
    cout << endl;
    switch(input){  //check input
        case 'w':
            hero.move(0,-1);
        break;
        case 's':
            hero.move(0,1);
        break;
        case 'a':
            hero.move(-1,0);
        break;
        case 'd':
            hero.move(1,0);
        break;
    }
    return 0;
}
```

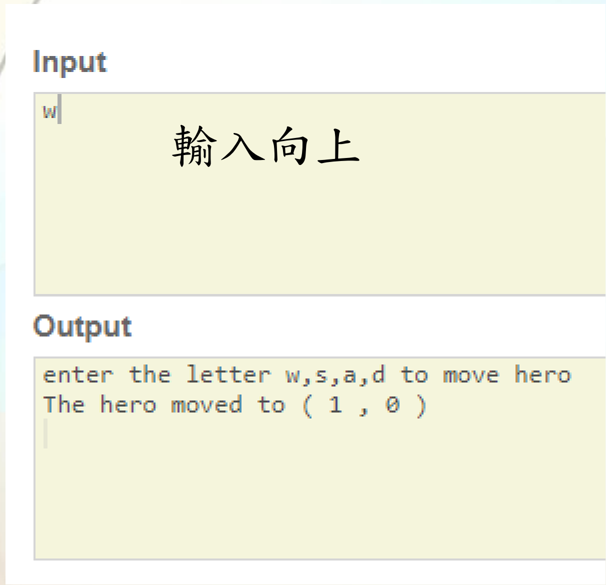
用cin取得輸入

用switch判斷輸入方向

heroMove:Output



主角位置



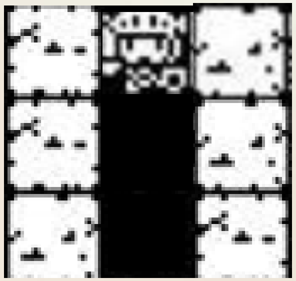
Input

```
w
```

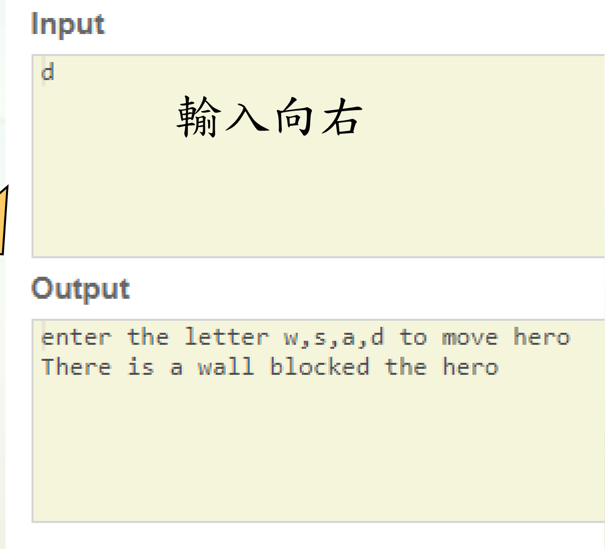
輸入向上

Output

```
enter the letter w,s,a,d to move hero
The hero moved to ( 1 , 0 )
```



主角向上移動。



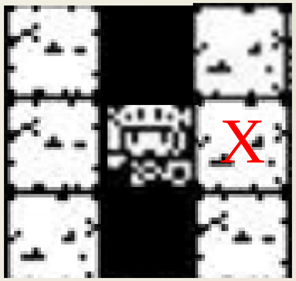
Input

```
d
```

輸入向右

Output

```
enter the letter w,s,a,d to move hero
There is a wall blocked the hero
```



主角右方被牆壁擋住，取消移動。

Declaring Void Functions

- Similar to functions returning a value
- Return type specified as "`void`"
- Example:
  - Function declaration/prototype:
    `void printStatus(int HP);`
    - Return-type is "`void`"
    - Nothing is returned

Declaring Void Functions

- Function definition:

```
void printStatus(int HP)
{
    if(HP > 0)
    {
          cout << “Hero is alive”<< endl;
    }
    else
    {
          cout << “Hero is dead”<< endl;
    }
}
```

- Notice: no return statement
    - Optional for void functions

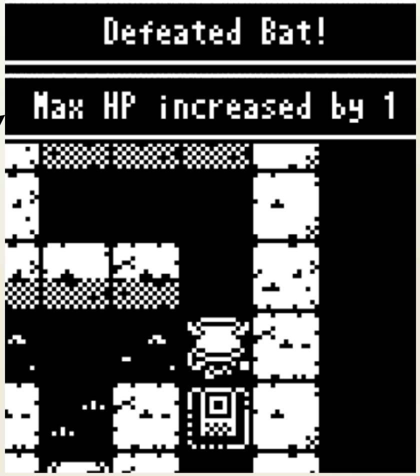
Calling Void Functions

- Same as calling predefined void functions
- From some other function, like `main()`:
  - `printStatus(HP);`
  - `printStatus(100);`
- Notice no assignment, since no value returned
- Actual arguments `(HP)`
  - Passed to function
  - Function is called to "do it’s job" with the data passed in

Function Example: `gainExp()`

- 當生物被英雄擊敗時會得到經驗值，若經驗值滿了提升等級與生命最大值。
- 每次提升等級時，下次升等所需的經驗值會越來越多。
- 相關變數:
  - 當前等級
  - 當前經驗值
  - 升級所需經驗值

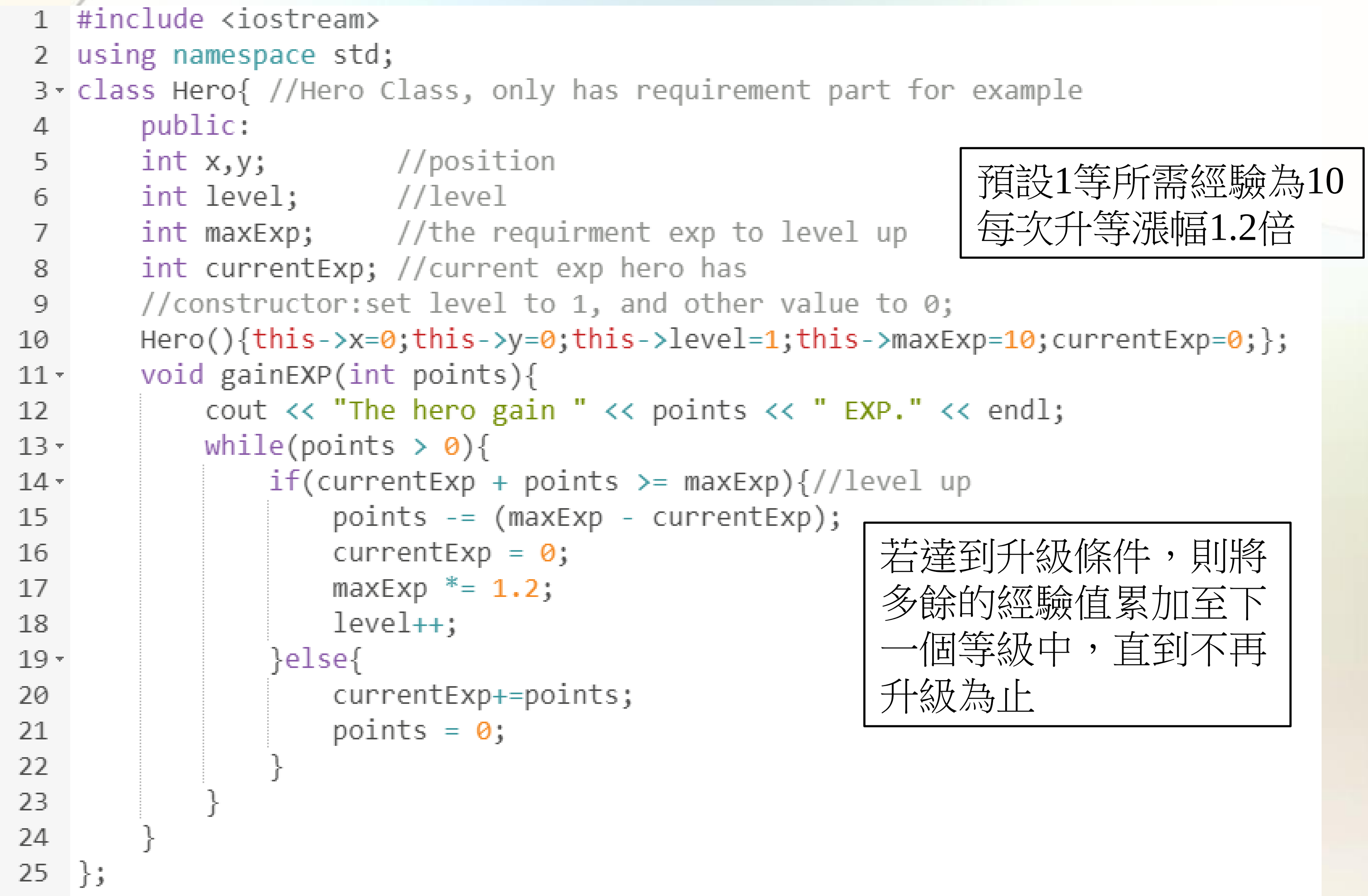
在遊戲中擊敗怪物會獲得經驗值並在升級後增加血量上限。



Defeated Bat!

Max HP increased by 1

gainExp : Example Code

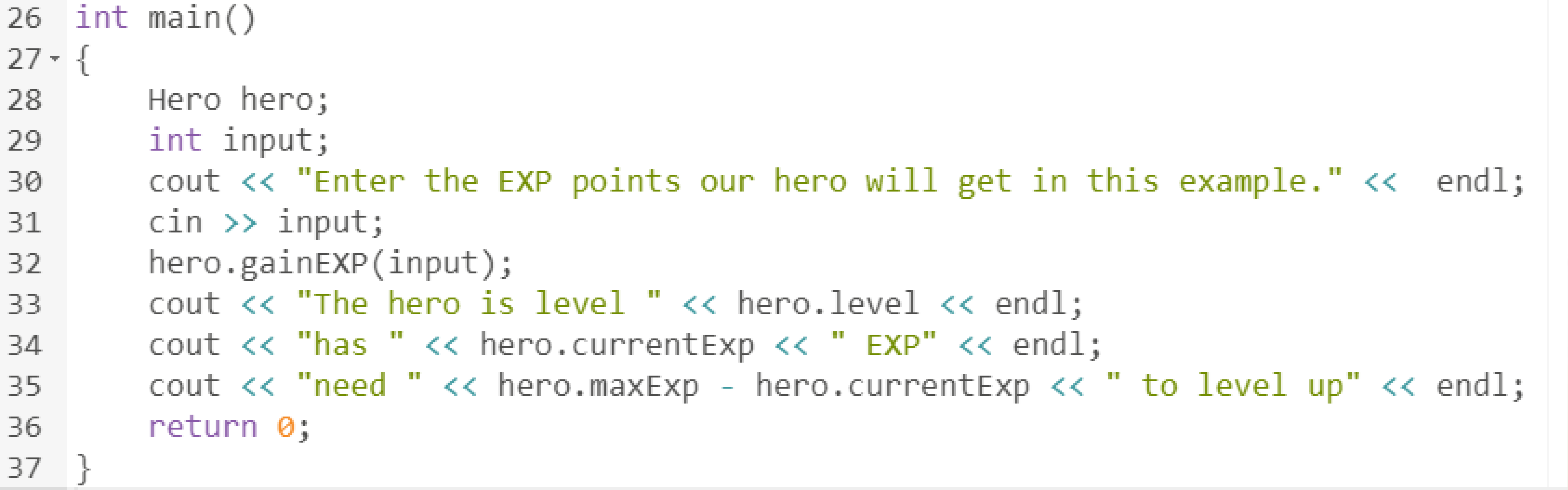


```cpp
#include <iostream>
using namespace std;
class Hero{ //Hero Class, only has requirement part for example
    public:
    int x,y;        //position
    int level;      //level
    int maxExp;     //the requirment exp to level up
    int currentExp; //current exp hero has
    //constructor:set level to 1, and other value to 0;
    Hero(){this->x=0;this->y=0;this->level=1;this->maxExp=10;currentExp=0;};
    void gainEXP(int points){
        cout << "The hero gain " << points << " EXP." << endl;
        while(points > 0){
            if(currentExp + points >= maxExp){//level up
                points -= (maxExp - currentExp);
                currentExp = 0;
                maxExp *= 1.2;
                level++;
            }else{
                currentExp+=points;
                points = 0;
            }
        }
    }
};
```

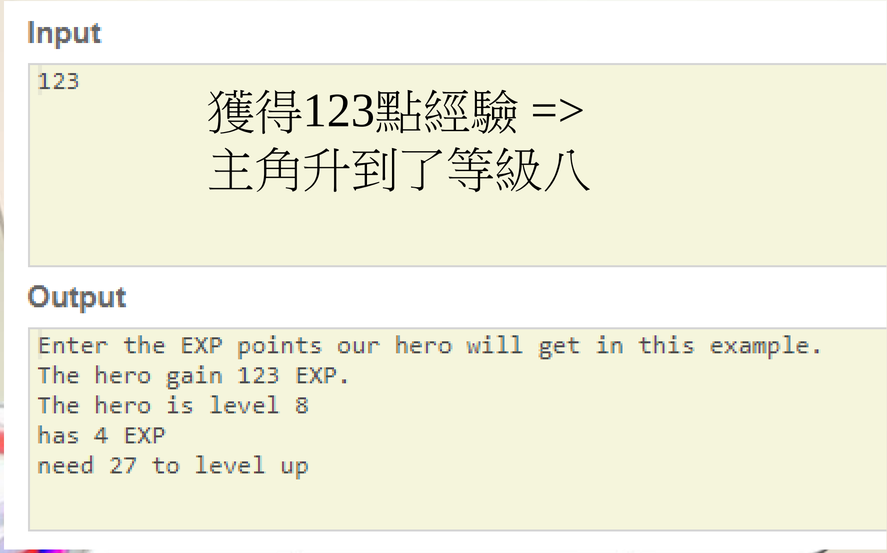
預設1等所需經驗為10
每次升等漲幅1.2倍

若達到升級條件，則將多餘的經驗值累加至下一個等級中，直到不再升級為止

gainExp:Output



```cpp
int main()
{
    Hero hero;
    int input;
    cout << "Enter the EXP points our hero will get in this example." <<  endl;
    cin >> input;
    hero.gainEXP(input);
    cout << "The hero is level " << hero.level << endl;
    cout << "has " << hero.currentExp << " EXP" << endl;
    cout << "need " << hero.maxExp - hero.currentExp << " to level up" << endl;
    return 0;
}
```



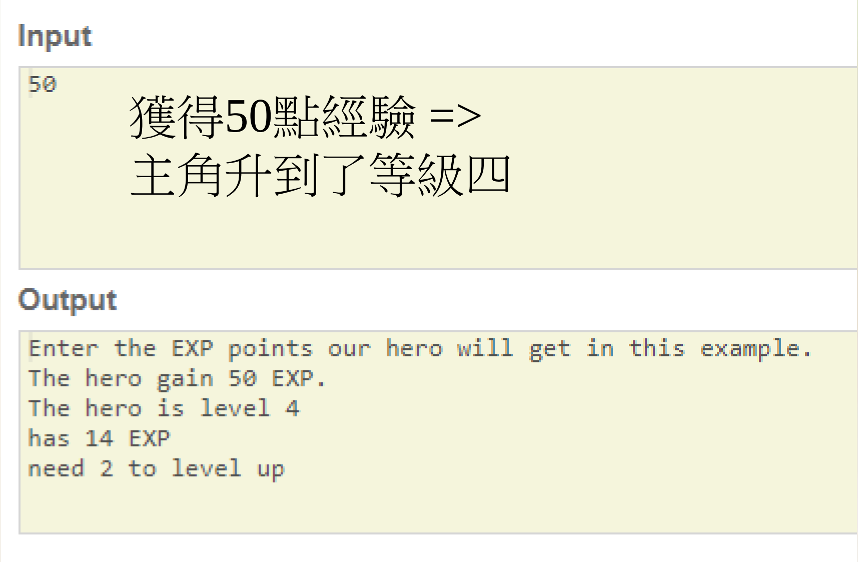
Input

```
123
```

獲得123點經驗 =>
主角升到了等級八

Output

```
Enter the EXP points our hero will get in this example.
The hero gain 123 EXP.
The hero is level 8
has 4 EXP
need 27 to level up
```



Input

```
50
```

獲得50點經驗 =>
主角升到了等級四

Output

```
Enter the EXP points our hero will get in this example.
The hero gain 50 EXP.
The hero is level 4
has 14 EXP
need 2 to level up
```

More on Return Statements

- Transfers control back to "calling" function
  - For return type other than void, MUST have `return` statement
  - Typically the LAST statement in function definition
- `return` statement optional for `void` functions
  - Closing } would implicitly return control from void function

Preconditions and Postconditions

- Similar to "**I-P-O**" discussion
- Comment function declaration:

```
int updateHealth(int HP, double distance);
//Precondition: distance is nonnegative and HP is larger
// than one
//Postcondition: New health after attack…
```

- Often called Inputs & Outputs

`main()`: "Special"

- Recall: `main()` IS a function
- "Special" in that:
  - One and only one function called `main()` will exist in a program
- Who calls `main()` ?
  - Operating system
  - Tradition holds it should have return statement
    - Value returned to "caller“ → Here: operating system
  - Should return "`int`" or "`void`"

Scope Rules

- Local variables
  - Declared inside body of given function
  - Available only within that function
- Can have variables with same names declared in different functions
  - Scope is local: "that function is it’s scope"
- Local variables preferred
  - Maintain individual control over data
  - Need to know basis
  - Functions should declare whatever local data needed to "do their job"

Procedural Abstraction

- Need to know "what" function does, not "how" it does it!
- Think "black box"
  - Device you know how to use, but not it’s method of operation
- Implement functions like black box
  - User of function only needs: declaration
  - Does NOT need function definition
    - Called Information Hiding
    - Hide details of "how" function does it’s job

Global Constants and Global Variables

- Declared "outside" function body
  - Global to all functions in that file
- Declared "inside" function body
  - Local to that function
- Global declarations typical for constants:

  -
    ```
    const double ATTACK = 20;
    ```
  - Declare globally so all functions have scope
- Global variables?
  - Possible, but SELDOM-USED
  - Dangerous: no control over usage!

Blocks

- Declare data inside compound statement
    - Called a "block"
    - Has "block-scope"
- Note: all function definitions are blocks!
    - This provides local "function-scope"
- Loop blocks:

```
for (int round = 1; round <= 10; round++)
{
    cout << “This is round ” << round << endl;
}
```

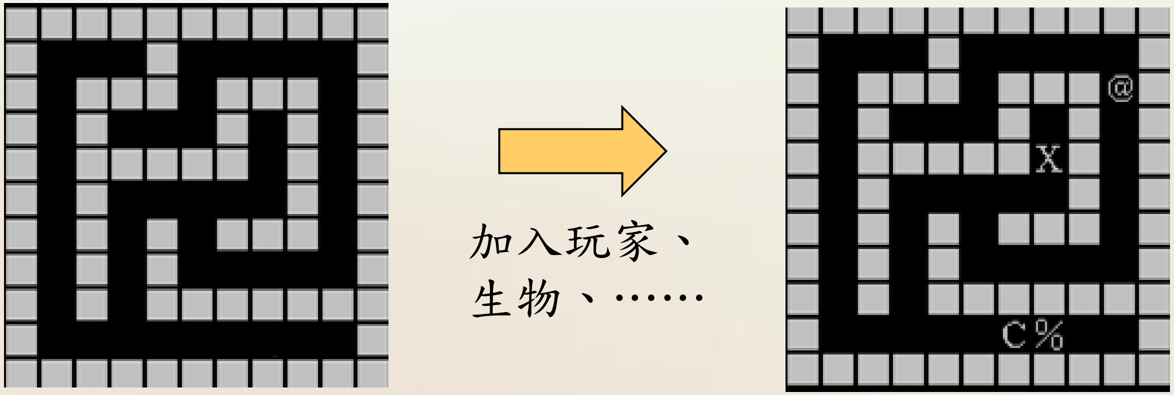
- Variable round has scope in loop body block only

Nested Scope

- Same name variables declared in multiple blocks
- Very legal; scope is "block-scope"
  - No ambiguity
  - Each name is distinct within its scope

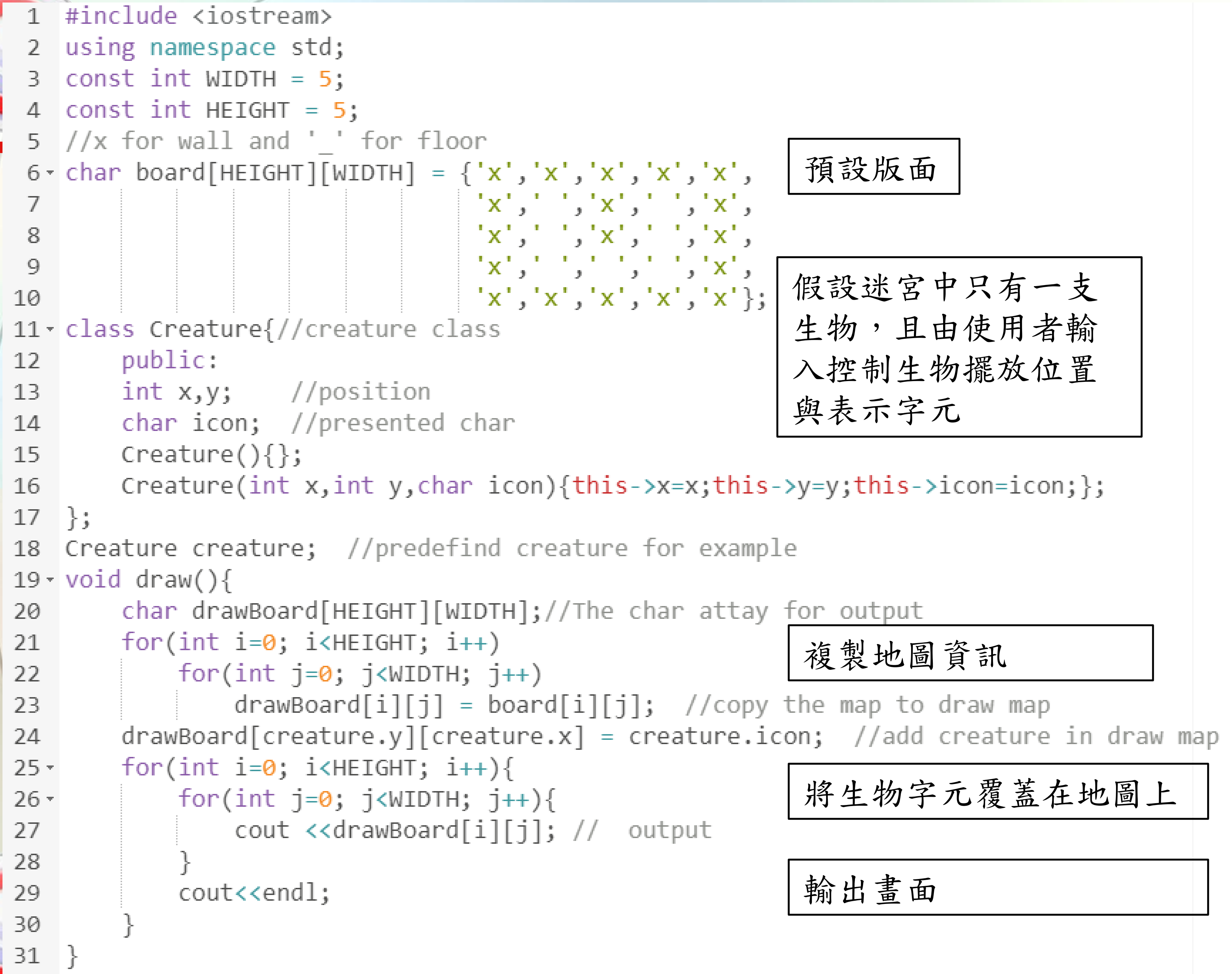
Block Example: `draw()`

- 當輸出遊戲畫面時，需要依序畫出場上的物件。
- 畫面顯示遊戲中分三層。
  - 地板與牆壁
  - 機關與道具，並覆蓋上一層顯示
  - 生物與主角，並覆蓋上一層顯示



加入玩家、
生物、……

遊戲中輸出的畫面以
字元陣列的方式輸出



```cpp
#include <iostream>
using namespace std;
const int WIDTH = 5;
const int HEIGHT = 5;
//x for wall and '_' for floor
char board[HEIGHT][WIDTH] = {'x','x','x','x','x',
                             'x',' ','x',' ','x',
                             'x',' ','x',' ','x',
                             'x',' ',' ',' ','x',
                             'x','x','x','x','x'};
class Creature{//creature class
    public:
    int x,y;    //position
    char icon;  //presented char
    Creature(){};
    Creature(int x,int y,char icon){this->x=x;this->y=y;this->icon=icon;};
};
Creature creature;  //predefind creature for example
void draw(){
    char drawBoard[HEIGHT][WIDTH];//The char attay for output
    for(int i=0; i<HEIGHT; i++)
        for(int j=0; j<WIDTH; j++)
            drawBoard[i][j] = board[i][j];  //copy the map to draw map
    drawBoard[creature.y][creature.x] = creature.icon;  //add creature in draw map
    for(int i=0; i<HEIGHT; i++){
        for(int j=0; j<WIDTH; j++){
            cout <<drawBoard[i][j]; //  output
        }
        cout<<endl;
    }
}
```

預設版面

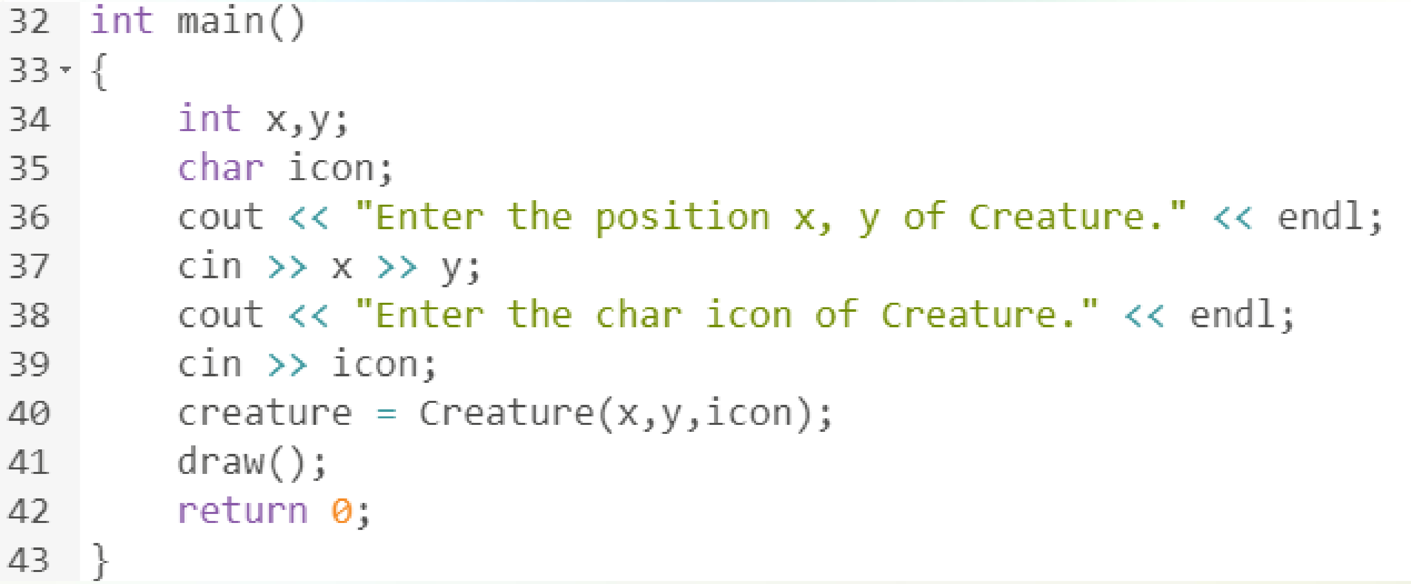
假設迷宮中只有一支生物，且由使用者輸入控制生物擺放位置與表示字元

複製地圖資訊

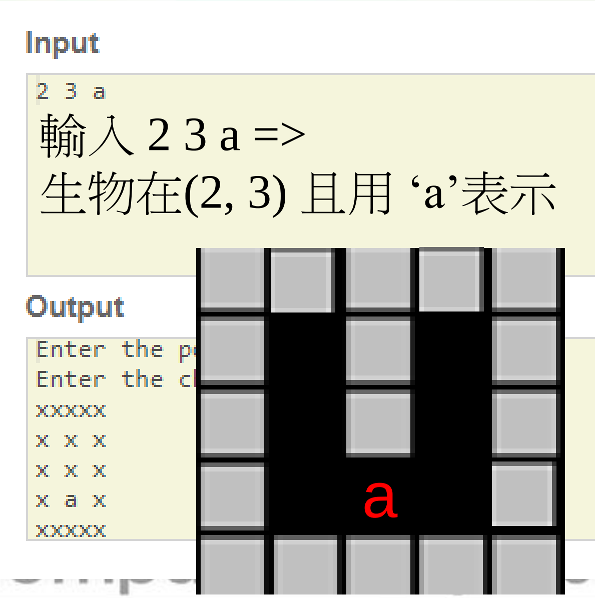
將生物字元覆蓋在地圖上

輸出畫面

draw() : Output



```
int main()
{
    int x,y;
    char icon;
    cout << "Enter the position x, y of Creature." << endl;
    cin >> x >> y;
    cout << "Enter the char icon of Creature." << endl;
    cin >> icon;
    creature = Creature(x,y,icon);
    draw();
    return 0;
}
```



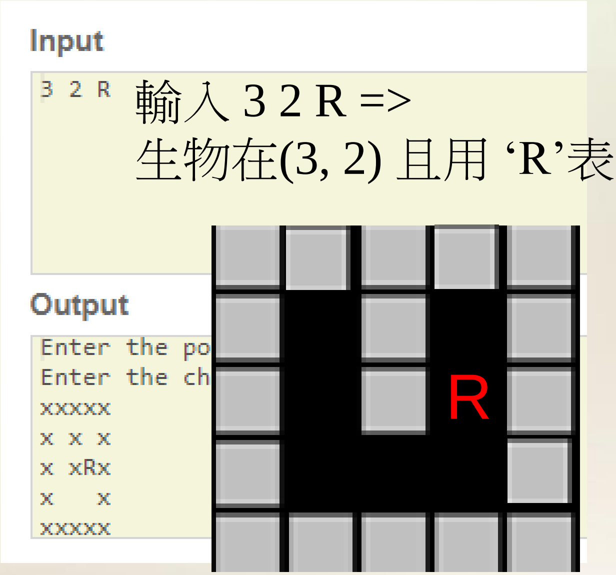
Input

```
2 3 a
```

輸入 2 3 a =>
生物在(2, 3) 且用 ‘a’表示

Output

```
Enter the p
Enter the c
xxxxx
x x x
x x x
x a x
xxxxx
```



Input

```
3 2 R
```

輸入 3 2 R =>
生物在(3, 2) 且用 ‘R’表示

Output

```
Enter the po
Enter the ch
xxxxx
x x x
x xRx
x   x
xxxxx
```

Chap03-02.cpp

```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
   vector<int> dynArrNums (3);

   dynArrNums[0] = 365;
   dynArrNums[1] = -421;
   dynArrNums[2]= 789;

   cout << "Number of integers in array: " << dynArrNums.size() << endl;
   cout << "Enter another number for the array" << endl;
   int anotherNum = 0;
   cin >> anotherNum;
   dynArrNums.push_back(anotherNum);

   cout << "Number of integers in array: " << dynArrNums.size() << endl;
   cout << "Last element in array: " << dynArrNums[ dynArrNums.size() - 1] <<
endl;

   return 0;
}
```

Note on Chap03-02.cpp

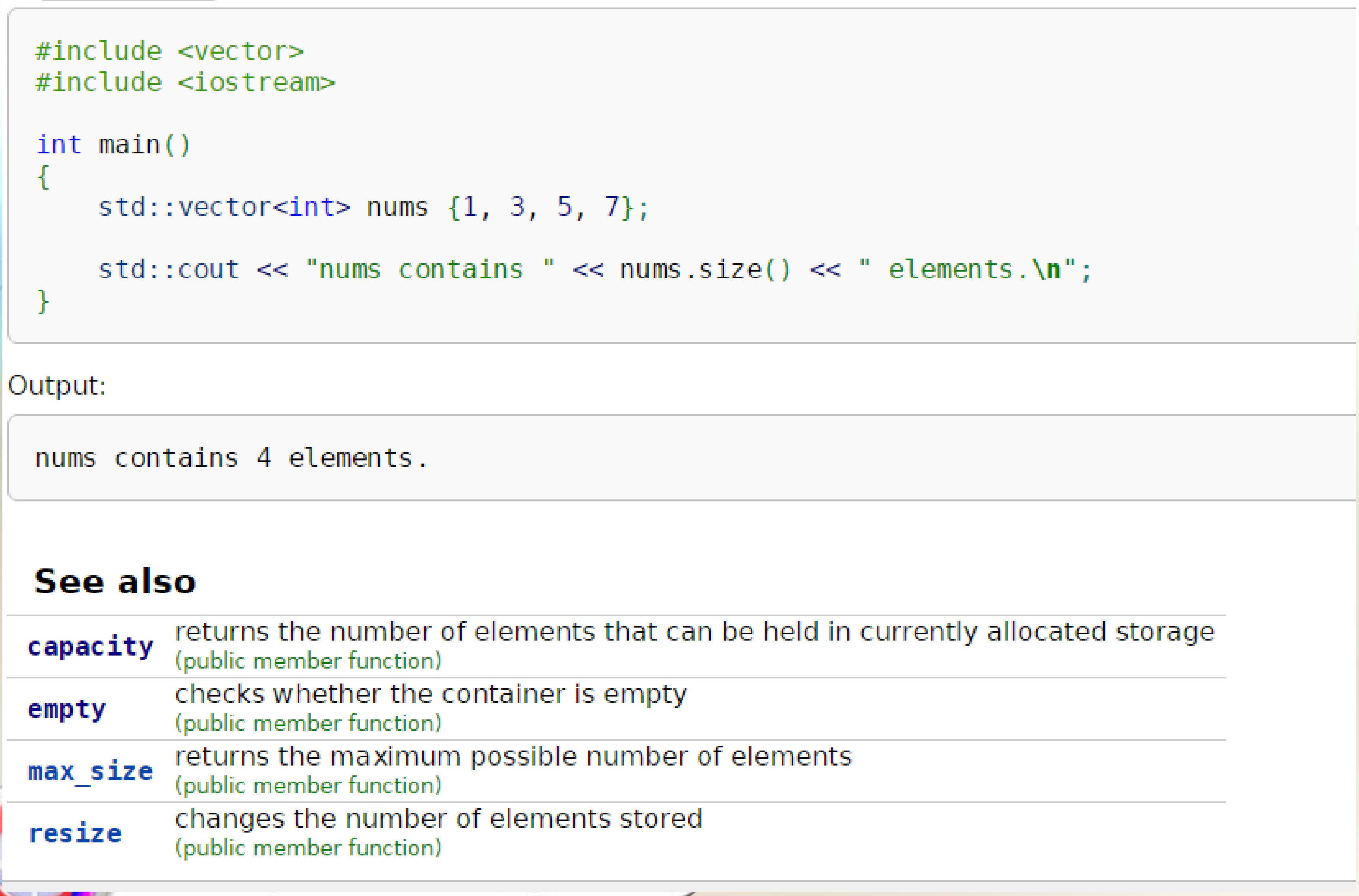
- Vector
- http://en.cppreference.com/w/cpp/container/vector

Examples of Vector

```cpp
#include <iostream>
#include <vector>

int main()
{
    // Create a vector containing integers
    std::vector<int> v = {7, 5, 16, 8};

    // Add two more integers to vector
    v.push_back(25);
    v.push_back(13);

    // Iterate and print values of vector
    for(int n : v) { // Range-based for loop to iterate through the array.
        std::cout << n << '\n';
    }
}
```

```
7
5
1
6
8
2
5
1
3
```

Examples of Vector



```
#include <vector>
#include <iostream>

int main()
{
    std::vector<int> nums {1, 3, 5, 7};

    std::cout << "nums contains " << nums.size() << " elements.\n";
}
```

Output:

```
nums contains 4 elements.
```

See also

capacity	returns the number of elements that can be held in currently allocated storage (public member function)
empty	checks whether the container is empty (public member function)
max_size	returns the maximum possible number of elements (public member function)
resize	changes the number of elements stored (public member function)

Summary 1

- Two kinds of functions:
  - "Return-a-value" and void functions
- Functions should be "black boxes"
  - Hide "how" details
  - Declare own local data
- Function declarations should self-document
  - Provide pre- & post-conditions in comments
  - Provide all "caller" needs for use

Summary 2

- Local data
  - Declared in function definition
- Global data
  - Declared above function definitions
  - OK for constants, not for variables
- Parameters/Arguments
  - Formal: In function declaration and definition
    - Placeholder for incoming data
  - Actual: In function call
    - Actual data passed to function