



Chapter 2

Flow of Control





想一想



- 要怎麼讓遊戲不斷更新及進行呢？要如何處理連續的讀入？
(CPU一秒跑好多次，要怎麼做才不會讓連續讀進方向指令)
- 要如何控制沒有限制的回合數的更新？如何控制並確保有固定或不固定數量的遊戲角色以及物件都有被讀取並更新狀態？
- 在遊戲中，遊戲有那些不同狀態？如何判斷現在遊戲是在什麼狀態？
 - 例如，結束遊戲。
- 如何記錄跟儲存遊戲狀態？(Week 8: IOStream)





想一想



- 在遊戲中會不斷loop每個回合，直到判斷遊戲結束。遊戲中也有許多地方會用到這兩類程式邏輯。
 - 那些遊戲機制會需要用判斷處理？用if 或 switch 哪個較合適？
 - 接收處理使用者操作輸入、判斷遊戲結束
 -
 - 那些遊戲機制會需要用loop處理？用for-loop 或 while-loop 哪個較合適？
 - 場景中生物群活動、在迷宮中配置多個陷阱或生物
 -
- fstream 可以讀檔也可以寫檔，也可以用來儲存遊戲進度。
 - 在迷宮遊戲中要儲存甚麼資料，才可以確保讀檔時遊戲內容會一樣？
 - 主角等級、迷宮等級、裝備名稱
 -
 - 遊戲儲存的資料，以甚麼方式輸出，以後讀檔才方便？





Learning Objectives

- Boolean Expressions
 - Building, Evaluating & Precedence Rules
- Branching Mechanisms
 - if-else
 - switch
 - Nesting if-else
- Loops
 - While, do-while, for
 - Nesting loops
- Introduction to File Input





Boolean Expressions:

Display 2.1 Comparison Operators

- Logical Operators
 - Logical AND (&&)
 - Logical OR (||)

Display 2.1 Comparison Operators

MATH SYMBOL	ENGLISH	C++ NOTATION	C++ SAMPLE	MATH EQUIVALENT
=	Equal to	==	<code>x + 7 == 2*y</code>	$x + 7 = 2y$
≠	Not equal to	!=	<code>ans != 'n'</code>	$ans \neq 'n'$
<	Less than	<	<code>count < m + 3</code>	$count < m + 3$
≤	Less than or equal to	<=	<code>time <= limit</code>	$time \leq limit$
>	Greater than	>	<code>time > limit</code>	$time > limit$
≥	Greater than or equal to	>=	<code>age >= 21</code>	$age \geq 21$





Evaluating Boolean Expressions

- Data type bool
 - Returns **true** or **false**
 - true, false are **predefined** library constants
- Truth tables
 - Display 2.2 next slide





Evaluating Boolean Expressions:

Display 2.2 Truth Tables

Display 2.2 Truth Tables

AND

<i>Exp_1</i>	<i>Exp_2</i>	<i>Exp_1</i> && <i>Exp_2</i>
true	true	true
true	false	false
false	true	false
false	false	false

OR

<i>Exp_1</i>	<i>Exp_2</i>	<i>Exp_1</i> <i>Exp_2</i>
true	true	true
true	false	true
false	true	true
false	false	false

NOT

<i>Exp</i>	!(<i>Exp</i>)
true	false
false	true



Display 2.3

Precedence of Operators (1 of 4)

Display 2.3 Precedence of Operators

::	Scope resolution operator	<i>Highest precedence (done first)</i>
.	Dot operator	
->	Member selection	
[]	Array indexing	
()	Function call	
++	Postfix increment operator (placed after the variable)	
--	Postfix decrement operator (placed after the variable)	
++	Prefix increment operator (placed before the variable)	
--	Prefix decrement operator (placed before the variable)	
!	Not	
-	Unary minus	
+	Unary plus	
*	Dereference	
&	Address of	
new	Create (allocate memory)	
delete	Destroy (deallocate)	
delete[]	Destroy array (deallocate)	
sizeof	Size of object	
()	Type cast	



Display 2.3

Precedence of Operators (2 of 4)

* / %	Multiply Divide Remainder (modulo)
+ -	Addition Subtraction
<< >>	Insertion operator (console output) Extraction operator (console input)



*Lower precedence
(done later)*





Display 2.3

Precedence of Operators (3 of 4)

Display 2.3 Precedence of Operators

All operators in part 2 are of lower precedence than those in part 1.

<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal
!=	Not equal
&&	And
	Or





Display 2.3

Precedence of Operators (4 of 4)

=	Assignment
+=	Add and assign
-=	Subtract and assign
*=	Multiply and assign
/=	Divide and assign
%=	Modulo and assign
? :	Conditional operator
throw	Throw an exception
,	Comma operator



*Lowest precedence
(done last)*





Precedence Examples

- Arithmetic before logical
 - `x + 1 > 2 || x + 1 < -3` means:
 - `(x + 1) > 2 || (x + 1) < -3`
- Short-circuit evaluation
 - `(x >= 0) && (y > 1)`
 - Be careful with increment operators!
 - `(x > 1) && (y++)`
- Integers as **boolean** values
 - All non-zero values \rightarrow **true**
 - Zero value \rightarrow **false**





Branching Mechanisms

- **if-else** statements
 - Choice of two alternate statements based on condition expression

- Example:

```
if (hrs > 40)
    grossPay = rate*40 + 1.5*rate*(hrs-40);
else
    grossPay = rate*hrs;
```





if-else Statement Syntax

- Formal syntax:

```
if (<boolean_expression>)
```

```
    <yes_statement>
```

```
else
```

```
    <no_statement>
```

- Note each alternative is only ONE statement!
- To have multiple statements execute in either branch → use compound statement





Compound/Block Statement

- Only "get" one statement **per branch**
- Must use compound statement { }
for multiples
 - Also called a "block" stmt
- Each block should have block statement
 - Even if just one statement
 - **Enhances readability**





Compound Statement in Action

- Note indenting in this example:

```
if (myScore > yourScore)
{
    cout << "I win!\n";
    wager = wager + 100;
}
else
{
    cout << "I wish these were golf
scores.\n";
    wager = 0;
}
```





Common Pitfalls

- Operator "=" vs. operator "=="

- One means "assignment" (=)
- One means "equality" (==)
- VERY different in C++!

- Example:

```
if (x = 12)    ←Note operator used!  
    Do_Something  
else  
    Do_Something_Else
```





The Optional else

- else clause is **optional**
 - If, in the false branch (else), you want "nothing" to happen, leave it out
 - Example:

```
if (sales >= minimum)
    salary = salary + bonus;
cout << "Salary = %" << salary;
```
 - Note: nothing to do for false condition, so there is no else clause!
 - Execution continues with `cout` statement





Nested Statements

- `if-else` statements contain smaller statements
 - Compound or simple statements (we've seen)
 - Can also contain any statement at all, including another `if-else` stmt!
 - Example:

```
if (speed > 55)
    if (speed > 80)
        cout << "You're really speeding!";
    else
        cout << "You're speeding.";
```
 - Note proper indenting!





Multiway if-else

- Not new, just different indenting
- Avoids "excessive" indenting
 - Syntax:

Multiway if-else Statement

SYNTAX

```
if (Boolean_Expression_1)
    Statement_1
else if (Boolean_Expression_2)
    Statement_2
    .
    .
    .
else if (Boolean_Expression_n)
    Statement_n
else
    Statement_For_All_Other_Possibilities
```



Multiway if-else Example

EXAMPLE

```
if ((temperature < -10) && (day == SUNDAY))  
    cout << "Stay home.";  
else if (temperature < -10) //and day != SUNDAY  
    cout << "Stay home, but call work.";  
else if (temperature <= 0) //and temperature >= -10  
    cout << "Dress warm.";  
else //temperature > 0  
    cout << "Work hard and play hard.";
```

The Boolean expressions are checked in order until the first true Boolean expression is encountered, and then the corresponding statement is executed. If none of the Boolean expressions is true, then the *Statement_For_All_Other_Possibilities* is executed.





if-else Example: 生物行為判斷

- 生物在行動前會判斷玩家加是否在附近，如果玩家在隔壁會攻擊，如果玩家在視線內會朝玩家移動，如果皆非則往隨機方向移動。

```
// 生物行為
```

```
if(range(creature, player) <= 1) // 生物在玩家隔壁
```

```
// 攻擊玩家
```

```
else if(canSee(creature, player)) // 生物看得到玩家
```

```
// 往玩家方向移動
```

```
else
```

```
// 隨機移動
```





The switch Statement

- A statement for controlling multiple branches
- Can do the same thing **with if statements** but sometimes switch is more convenient
- Uses controlling expression **which returns bool data** type (true or false)
- Syntax:
 - Next slide





The switch Statement Syntax

switch Statement

SYNTAX

```
switch (Controlling_Expression)
{
    case Constant_1:
        Statement_Sequence_1
        break;
    case Constant_2:
        Statement_Sequence_2
        break;
        .
        .
        .
    case Constant_n:
        Statement_Sequence_n
        break;
    default:
        Default_Statement_Sequence
}
```

*You need not place a **break** statement in each case. If you omit a **break**, that case continues until a **break** (or the end of the **switch** statement) is reached.*

The controlling expression must be integral! This includes char.





The switch Statement in Action

EXAMPLE

```
int vehicleClass;
double toll;
cout << "Enter vehicle class: ";
cin >> vehicleClass;

switch (vehicleClass)
{
    case 1:
        cout << "Passenger car.";
        toll = 0.50;
        break;
    case 2:
        cout << "Bus.";
        toll = 1.50;
        break;
    case 3:
        cout << "Truck.";
        toll = 2.00;
        break;
    default:
        cout << "Unknown vehicle class!";
}
```

*If you forget this break,
then passenger cars will
pay \$1.50.*



The switch: multiple case labels

- Execution "falls thru" until break

- switch provides a "point of entry"

- Example:

```
case 'A':  
case 'a':  
    cout << "Excellent: you got an "A"!\n";  
    break;  
case 'B':  
case 'b':  
    cout << "Good: you got a "B"!\n";  
    break;
```

- Note multiple labels provide same "entry"





switch Pitfalls/Tip

- Forgetting the `break`;
 - No compiler error
 - Execution simply "falls thru" other cases until `break`;
- Biggest use: **MENUs**
 - Provides clearer "big-picture" view
 - Shows menu structure **effectively**
 - Each branch is one menu choice





switch Menu Example

- Switch stmt "perfect" for menus:

```
switch (response)
{
    case 1:
        // Execute menu option 1
        break;
    case 2:
        // Execute menu option 2
        break;
    case 3:
        // Execute menu option 3
        break;
    default:
        cout << "Please enter valid response.";
}
```





Switch Example: 輸入判斷

- 玩家在遊戲中可以輸入方向(W、S、A、D)控制移動方向。

```
char input; cin >> input;
```

```
switch (input)
```

```
{
```

```
    case 'w': // 玩家向上移動
```

```
        break;
```

```
    case 's': // 玩家向下移動
```

```
        break;
```

```
    case 'a': // 玩家向左移動
```

```
        break;
```

```
    case 'd': // 玩家向右移動
```

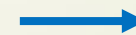
```
        break;
```

```
    default:
```

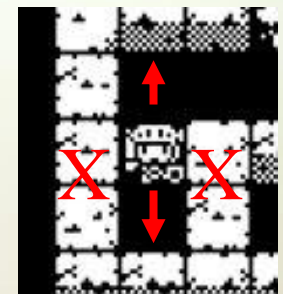
```
        cout << "Please enter valid input.";
```

```
}
```

玩家輸入方向



角色移動





Conditional Operator

- Also called "ternary operator"

- Allows embedded conditional in expression
- Essentially "shorthand if-else" operator

- Example:

```
if (creatureAttack > heroArmor)
    heroArmor = 0;
else
    heroArmor -= creatureAttack;
```

- Can be written:

```
heroArmor = (creatureAttack > heroArmor)
            ? 0 : heroArmor - creatureAttack;
```

- "?" and ":" form this "ternary" operator





Loops

- 3 Types of loops in C++
 - while
 - Most flexible
 - No "restrictions"
 - do-while
 - Least flexible
 - Always executes loop body at least once
 - for
 - Natural "counting" loop





while Loops Syntax

Syntax for while and do-while Statements

A while STATEMENT WITH A SINGLE STATEMENT BODY

```
while (Boolean_Expression)  
    Statement
```

A while STATEMENT WITH A MULTISTatement BODY

```
while (Boolean_Expression)  
{  
    Statement_1  
    Statement_2  
    .  
    .  
    .  
    Statement_Last  
}
```





while Example

- Consider:

```
count = 0;           // Initialization
while (count < 3)    // Loop Condition
{
    cout << "Hi "; // Loop Body
    count++;        // Update expression
}
```

- Loop body executes how many times?





while Loop: 遊戲回合執行

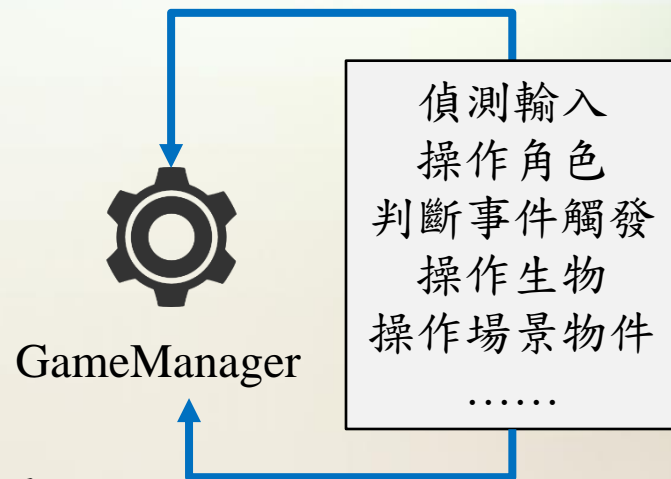
- 遊戲在執行時會不斷loop每個回合，直到判斷遊戲結束。

```
bool gameOver = false;  
while (!gameOver) //直到遊戲結束，一直重複下個回合  
{
```

```
    //偵測輸入  
    //操作角色  
    //判斷事件觸發  
    //操作生物  
    //操作場景物件  
    //.....
```

```
    if (player.health <= 0) //血量歸零  
        gameOver = true;
```

```
}
```





do-while Loop Syntax

A do-while STATEMENT WITH A SINGLE-STATEMENT BODY

```
do  
    Statement  
while (Boolean_Expression);
```

A do-while STATEMENT WITH A MULTISTatement BODY

```
do  
{  
    Statement_1  
    Statement_2  
    .  
    .  
    .  
    Statement_Last  
} while (Boolean_Expression);
```

*Do not forget
the final
semicolon.*





do-while Loop Example

- ```
count = 0; // Initialization
do
{
 cout << "Hi "; // Loop Body
 count++; // Update expression
} while (count < 3); // Loop Condition
```
- Loop body executes **how many times**?
- do-while loops always execute body at least once!







# while vs. do-while

- Very similar, but...
  - One important difference
    - Issue is "WHEN" boolean expression is checked
    - while: checks BEFORE body is executed
    - do-while: checked AFTER body is executed
- After this difference, they're essentially identical!
- while is more common, due to its ultimate "flexibility"





# Comma Operator

- Evaluate list of expressions, **returning value** of the **last** expression
- Most often used in a **for-loop**
- Example:  

```
first = (first = 2, second = first + 1);
```

  - **first** gets assigned the value **3**
  - second gets assigned the value 3
- No guarantee what order expressions will be evaluated.





# for Loop Syntax

```
for (Init_Action; Bool_Exp; Update_Action)
 Body_Statement
```

- Like if-else, Body\_Statement can be a block statement
  - Much more typical





# for Example

- ```
for (count=0; count<3; count++)  
{  
    cout << "Hi "; // Loop Body  
}
```
- How many times does loop body execute?
- Initialization, loop condition and update all "built into" the for-loop structure!
- A natural "counting" loop





for: 生物行為處理

- 在遊戲中，複數的生物群是用陣列方式儲存管理，並用index編號辨別。
- 當遊戲需要處理不同生物的行為時可利用for loop 管理。

// 用for loop 管理生物群

```
for (count=0; count<creature.length; count++)
```

```
{
```

```
    // 生物行為，處理移動攻擊等等
```

```
    creature[count].active();
```

```
}
```



生物1號



生物2號

⋮



生物(length - 1)號





Loop Issues

- Loop's condition expression **can be ANY** boolean expression
- Examples:

```
while (time < stopTime && health!=0)
{
    // Do something
}
```

```
for (time = 0; time < stopTime && health != 0)
{
    // Do something
}
```





Loop Pitfalls: Misplaced ;

- Watch the **misplaced ;** (semicolon)
 - Example:

```
while (health != 0) ;←  
{  
    cout << "Enter direction: ";  
    cin >> direction;  
}
```
 - Notice the ";" after the while condition!
- Result here: INFINITE LOOP!





Loop Pitfalls: Infinite Loops

- Loop condition **must evaluate to false** at some iteration through loop
 - If not → **infinite loop**.
 - Example:

```
while (1)
{
    cout << "Current time: " << time << " secs";
}
```
 - A perfectly legal C++ loop → always infinite!
- Infinite loops can be desirable
 - e.g., "Embedded Systems"





The break and continue Statements

- Flow of Control
 - Recall how loops provide "graceful" and clear flow of control in and out
 - In RARE instances, can alter natural flow
- **break;**
 - Forces loop to exit immediately.
- **continue;**
 - Skips rest of loop body
- These statements violate natural flow
 - Only used when absolutely necessary!





Nested Loops

- Recall: ANY valid C++ statements can be inside body of loop
- This includes additional loop statements!
 - Called "nested loops"

- Requires careful indenting:

```
for (outer=0; outer<5; outer++)  
    for (inner=7; inner>2; inner--)  
        cout << outer << inner;
```

- Notice no { } since each body is one statement
- Good style dictates we use { } anyway





Introduction to File Input

- We can use `cin` to read from a file in a manner very similar to reading **from the keyboard**
- Only an introduction is given here, more details are in chapter 12
 - Just enough so **you can read from text files** and process larger amounts of data that would be too much work to type in





Opening a Text File

- Add at the top

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

- You can then declare an **input stream** just as you would declare any other variable.

```
ifstream inputStream;
```

- Next you must connect the inputStream variable to a text file on the disk.

```
inputStream.open("level1.txt");
```

- The “filename.txt” is the pathname to a text file or a file in the current directory





Reading from a Text File

- Use

```
istream >> HP;
```

- The result is the same as using `cin >> HP` except the input is coming from the text file and not the keyboard
- When done with the file close it with

```
istream.close();
```





File Input Example (1 of 2)

- Consider a text file named *player.txt* with the following text

Display 2.10 Sample Text File, `player.txt`, to Store a Player's High Score and Name

100510

Gordon Freeman





File Input Example (2 of 2)

Display 2.11 Program to Read the Text File in Display 2.10

```
1  #include <iostream>
2  #include <fstream>
3  #include <string>

4  using namespace std;
5  int main( )
6  {
7      string firstName, lastName;
8      int score;
9      fstream inputStream;

10     inputStream.open("player.txt");

11     inputStream >> score;
12     inputStream >> firstName >> lastName;

13     cout << "Name: " << firstName << " "
14          << lastName << endl;
15     cout << "Score: " << score << endl;
16     inputStream.close();

17     return 0;
18 }
```

Sample Dialogue

Name: Gordon Freeman
Score: 100510



Notes on Display 2.11

- **fstream**

- <https://msdn.microsoft.com/zh-tw/library/6z061fh0.aspx>
- basic_fstream: <https://msdn.microsoft.com/zh-tw/library/a33ahe62.aspx>

- Member function

- .close() :
- .open()
- http://en.cppreference.com/w/cpp/io/basic_fstream

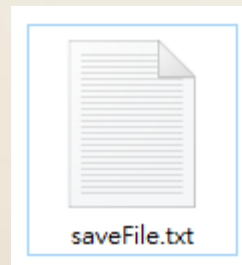




ifstream: 遊戲讀取存檔

// 用ifstream讀取原先已自錄儲存之原始位置狀態

```
void readFile() {  
    std::ifstream file;  
    file.open("savefile.txt");  
    int heroX, heroY, triggerX, triggerY, creatureX, creatureY;  
    std::string buffer;  
    file >> heroX >> heroY;  
    file >> triggerX >> triggerY;  
    file >> creatureX >> creatureY;  
    gCreature.setPos(creatureX, creatureY);  
    gTrigger.setPos(triggerX, triggerY);  
    gHero.setPos(heroX, heroY);  
    setupBoard();  
    file.close();  
}
```



(x, y)



(x, y)



(x, y)



ofstream: 遊戲讀取存檔

// 用ofstream記錄現今遊戲狀態

```
void saveFile() {  
    std::ofstream outfile("saveFile.txt");  
    outfile << hero.x << ' ' << hero.y << std::endl;  
    outfile << trig.x << ' ' << trig.y << std::endl;  
    outfile << cre.x << ' ' << cre.y << std::endl;  
    outfile.close();  
}
```



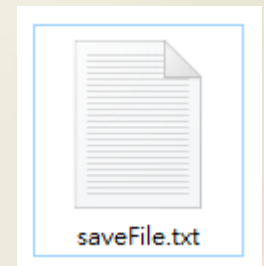
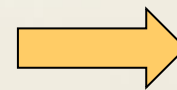
(x, y)



(x, y)



(x, y)





Summary 1

- Boolean expressions
 - Similar to arithmetic → results in true or false
- C++ branching statements
 - `if-else`, `switch`
 - `switch` statement great for menus
- C++ loop statements
 - `while`
 - `do-while`
 - `for`





Summary 2

- `do-while` loops
 - Always execute their loop body at least once
- `for`-loop
 - A natural "counting" loop
- Loops can be exited early
 - `break` statement
 - `continue` statement
 - Usage restricted for style purposes
- Reading from a text file is similar to reading from `cin`

