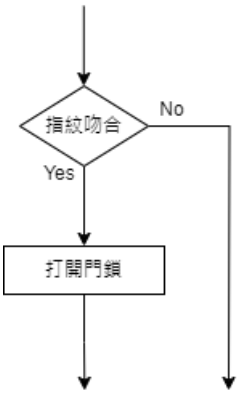
03-選擇結構

3.1 if ... else ...

每一行程式碼都會執行到？

我們寫的每一行程式碼都有用嗎？當然有用。那每一行都會被執行嗎？這要看情況。

之前我們寫的程式，會從 `main()` 函數的第一行開始一行一行依序執行下去，直到程式結束。但在真實世界運作的程式是要有彈性的，例如：指紋鎖必須要在使用者指紋與內部設定吻合時，才會開鎖，否則什麼都不做。也就是這樣一個結構：



接下來我們我們以「計算絕對值」為例，來看看這種結構。

if 敘述

練習：絕對值

讀入使用者輸入的整數 a，計算並輸出其絕對值 $|a|$。

絕對值表示數線上原點到該數值的距離，所以若 $a\ge 0$ 則 a 的值就是其絕對值，否則將 a 的值乘上 -1 才是其絕對值。

```
#include <iostream>

using namespace std;

int main()
{
    int a;

    cin >> a;

    if(a<0)
    {
        a = -1*a;
    }

    cout << "|a|=" << a << endl;

    return 0;
}
```

在這裡我們使用到 if 敘述，它的語法如下：

```
if( 條件判斷式 )
{
    條件成立時要做的事
}
```

if 後面的小括號裡是個 **條件判斷式**，它的運算結果必需是 **布林值(boolean)**。

如果條件判斷式的運算結果為 **true** 則執行接下來那組大括號內的程式碼，否則就略過那整個大括號的內容。

關鍵往往在於你是否能找到一個合宜的條件判斷式，來抓到你要的狀態。

練習：判斷奇數(odd number)

讀入使用者輸入的整數 a，若其為奇數，輸出 "奇數" 否則什麼都不做。

我們要怎麼判定 a 是奇數呢？只要把它除以 2，看餘數是不是 1 就知道了。

```
int a;
cin >> a;

if(a%2==1)
{
    cout << "奇數" << endl;
}
```

if ... else ...

如果我們在條件判斷式成立時要做「工作A」，不成立時要做「工作B」，該怎麼辦呢？

這種 2 條分支的狀況，可以使用 **if ... else ...** 敘述，語法如下。

```
if( 條件判斷式 )
{
    條件成立時要做的事
}
else{
    條件不成立時要做的事
}
```

可以把它想成是「**如果** (...) **就做** {...} **否則做** {...}」

練習：判斷奇、偶數

讀入使用者輸入的整數 a，若其為奇數，輸出 "奇數" 否則輸出 "偶數"。

```
int a;
cin >> a;

if(a%2==1)
{
    cout << "奇數" << endl;
}
else
{
    cout << "偶數" << endl;
```

```
}
```

if ... else if ... else

狀況再複雜一點，如果分支多於 2 條呢？例如：判斷 a 是「正數」、「負數」還是「0」

這時我們需要加入 **else if**，來做到 「**如果** (...) **就做** {...} **否則如果** (...) **就做** {...} **否則做** ...」

```
if( 條件判斷式1 )
{
    條件1成立時要做的事
}
else if( 條件判斷式2 )
{
    條件2成立時要做的事
}
else
{
    前面條件都不成立時要做的事
}
```

練習：正、負數與零的判斷

讀入使用者輸入的整數 a ，輸出其為 "正數"、"負數" 或 "零"。

```
int a;

cin >> a;

if(a>0)
{
    cout << "正數" << endl;
}
else if(a<0)
{
    cout << "負數" << endl;
}
else
{
    cout << "零" << endl;
}
```

分支大於 3 條時，只要重覆多個 **else if** 即可。

練習：分數轉換為等第

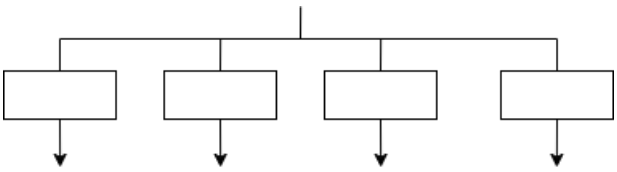
讀取使用者輸入的成績，輸出其相應的等第，轉換參考表如下。

分數	等第
90 ~	A
80 ~ 89	B
70 ~ 79	C
60 ~ 69	D
~ 59	E

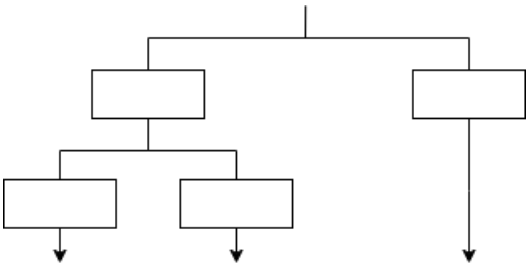
```
int score;

cin >> score;

if(score>=90)
{
    cout << "A" << endl;
}
else if(score>=80)
{
    cout << "B" << endl;
}
else if(score>=70)
{
    cout << "C" << endl;
}
else if(score>=60)
{
    cout << "D" << endl;
}
else
{
    cout << "E" << endl;
}
```

巢狀(nested)/多層 結構

目前我們遇到的選擇結構是像這樣 **單層多分支** 。



但是也有像這樣 **多層多分支** 的選擇結構。



練習：是否需服兵役

由使用者輸入性別、年齡，只有男生且年齡大於等於 20 歲才需要服兵役。

只有 2 個檢查條件都成立，才會被判定需當兵。

```
string gender;
cout << "性別(男, 女):";
cin >> gender;

int age;
cout << "年齡:";
cin >> age;

if(gender=="男")
{
    cout << "你是男生，";
    if(age>=20)
    {
```

```
        cout << "需要當兵" << endl;
    }
   else
    {
        cout << "但是年紀太小，還不用當兵" << endl;
    }
}
else
{
    cout << "你是女生，不用當兵" << endl;
}
```

在這裡我們用到了如下的巢狀結構。

```
if( 條件判斷式1 )
{
    條件 1 成立時要做的事
    if( 條件判斷式2 )
    {
        條件 1, 2 都成立時要做的事
    }
    else
    {
        條件 1 成立、2 不成立時要做的事
    }
}
else
{
    條件 1 不成立時要做的事
}
```

一組大括號包起來的部分，我們稱為一個程式區塊(block)。每個區塊內都可以再塞進其他的區塊。

縮排(indent)

如前所述我們可以有各式各樣很多層次的程式碼。當層次一多起來，程式碼就會開始亂，連自己的的東西都不容易看懂。

所以在撰寫 C++ 程式時，我們都會遵守一個規範，在出現大括號時，裡面的程式碼就會縮一層(4個空白字元或是一個[tab])，這樣可以讓程式碼的層次一目了然。

雖然不縮排，程式也能執行，但是你的伙伴會看不懂你在寫什麼，也不會想看你的程式。

```
string genger;
cout << "性別(男, 女):";
cin >> gender;

int age;
cout << "年齡:";
cin >> age;

if(gender=="男")
{
    cout << "你是男生，";
    if(age>=20)
    {
        cout << "需要當兵" << endl;
    }
    else
    {
        cout << "但是年紀太小，還不用當兵" << endl;
    }
}
else
{
    cout << "你是女生，不用當兵" << endl;
}
```

3.2 關於 if 敘述大括號的使用

內容只有一行時可以省略大括號

`if...else if...else` 的大括號內如果只有一行時，可以省略大括號。

所以前面範例練習的內容可以寫成這樣。

練習：絕對值

```
#include <iostream>

using namespace std;

int main()
{
    int a;

    cin >> a;

    if(a<0)
        a = -1*a;

    cout << "|a|=" << a << endl;

    return 0;
}
```

練習：判斷奇、偶數

```
int a;
cin >> a;

if(a%2==1)
    cout << "奇數" << endl;
else
    cout << "偶數" << endl;
```

但並不建議同學們這樣做，因為這樣有時會難以閱讀而造成意外的錯誤，不如老老實實的都加上大括號。

例如：下面這段程式碼就很難清楚理解，事實上是有錯誤的。

練習：是否需服兵役

```
string gender;
cout << "性別(男, 女):";
cin >> gender;

int age;
cout << "年齡:";
cin >> age;

if(gender=="男")
    cout << "你是男生，";
    if(age>=20)
        cout << "需要當兵" << endl;
   else
        cout << "但是年紀太小，還不用當兵" << endl;
else
    cout << "你是女生，不用當兵" << endl;
```

兩種常見的大括號使用風格

風格一：左右括號都單獨佔一行

```
int a;
cin >> a;

if(a%2==1)
{
    cout << "奇數" << endl;
}
else
{
    cout << "偶數" << endl;
}
```

風格二：左括號放在行末

```
int a;
cin >> a;

if(a%2==1) {
    cout << "奇數" << endl;
}
else {
    cout << "偶數" << endl;
}
```

這兩種撰寫風格都很常見，同學們可以自行選擇。唯一的提醒就是 **務必要正確的縮排**。

3.3 複合條件判斷式

搭配使用邏輯運算子

底下是一個典型的帳密驗證程式片斷。

```
string id, password;

cin >> id;
cin >> password;

if(id=="admin")
{
    if(password=="123456")
    {
        cout << "登入成功" << endl;
    }
    else
    {
        cout << "登入失敗" << endl;
    }
}
else
{
    cout << "登入失敗" << endl;
}
```

因為「帳號正確」、「密碼正確」兩者皆需成立，所以使用了二層 `if...else` 敘述，看起來很累贅。

And 邏輯運算子 &&

「帳號正確」**而且**「密碼正確」可以這樣表示。

```
string id, password;

cin >> id;
cin >> password;

if(id=="admin" && password=="123456")
{
    cout << "登入成功" << endl;
}
else
{
    cout << "登入失敗" << endl;
}
```

邏輯運算子可以對 **布林值(boolean)** 進行運算。

我們把 `id=="admin"` 視為**條件A**，`password=="123456` 視為**條件B**

條件A 的運算結果有兩種可能 true, false。條件B 也是一樣。所以 A and B 有四種可能狀況，表列如下。

A	B	A && B
false	false	false
false	true	false
true	false	false
true	true	true

這種表叫做 **「真值表(truth table)」** ，可以表示某個邏輯運算的各種狀態和運算結果，我們也很常用 0 表示 false，用 1 表示true。

A	B	A && B
0	0	0
0	1	0
1	0	0
1	1	1

由這張 And 的真值表可以看出只有在兩個條件都是 true 的情況下 A and B 的邏輯運算結練才會是 true，其他狀況都是 false。

Or 邏輯運算子 ||

Or 的真值表如下，只要 A, B 其中之一為 true，運算結果就會是 true。

A	B	A || B
0	0	0
0	1	1
1	0	1
1	1	1

如果獲得獎學金的資格是「國文成績85(含)以上」或「英文成績80(含)以上」，則我們可以用 Or 運算子這樣判斷個案是否可以領取獎學金。

```
if(chi>=85 || eng>=80)
{
    cout << "符合領取獎學金資格" << endl;
}
```

Nor 邏輯運算子 !

And 和 Or 運算子左右各有一個運算元，所以我們稱之為 **二元運算子**。

Not 只有右邊一個運算元，所以它是 **一元運算子**。Not 的運算結果是把它後面那個運算元的真值反轉，也就是 true 變成 false，false 變成 true。

A	!A
0	1
1	0

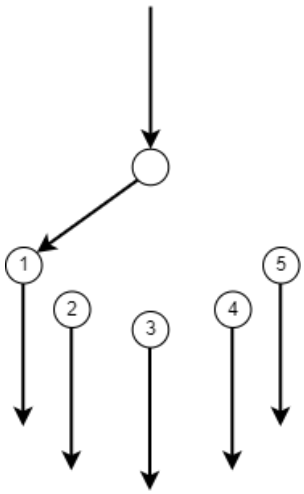
我們可以這樣來過濾 n「不是 4 的倍數」

```
if(!(n%4==0))
{
    cout << n << "不是 4 的倍數" << endl;
}
```

3.4 switch ... case

另一種「多分支」選擇結構

當你遇到像這樣的多分支選擇結構時，可以用 `if...else if...else` 來解決。



例如：一個像這樣的選單功能

```
#include <iostream>

using namespace std;

int main()
{
    cout << "(1) 提款" << endl;
    cout << "(2) 存款" << endl;
    cout << "(3) 查詢帳上餘額" << endl;
    cout << "(0) 結束" << endl;
    cout << "請選擇服務項目(0-3):";

    int i;
    cin >> i;

    if(i==1) {
        cout << "進行提款作業中..." << endl;
        cout << "提款作業完成!" << endl;
    }
    else if(i==2) {
        cout << "進行存款作業中..." << endl;
        cout << "存款作業完成!" << endl;
    }
    else if(i==3) {
        cout << "進行查詢作業中..." << endl;
        cout << "查詢作業完成!" << endl;
    }
    else if(i==0) {
        cout << "結束服務" << endl;
    }
    else {
        cout << "無此項目" << endl;
    }

    return 0;
}
```

除此之外，在 C++ 裡還有另一種 `switch ... case ...` 敘述，非常適合這種選單型的多分支結構。

它的基本語法如下：

```
switch (運算式)
{
    case 常數運算式1:
        ...
        break;
    case 常數運算式2:
        ...
        break;
    ...
    default:
        ...
}
```

- **switch** 後面括號中的運算式，其運算結果必需是 *整數* 或是 *字元*。
- 程式執行到 **switch** 時，會把小括號內運算式的運算結果拿來依序比對 **case** 後面的常數運算式(可能是 *整數* 或是 *字元*)。如果發現符合就跳到那個 **case** 的下一行開始執行，直到遇到 **break** 離開 switch 程式區塊。
- 如果 switch 的程式區塊中有 **default** 關鍵字，則當所有的 case 都不相符時，將由 default 處開始執行。

前面的選單程式，可以改寫成這樣。

```
#include <iostream>

using namespace std;

int main()
{
    cout << "(1) 提款" << endl;
    cout << "(2) 存款" << endl;
    cout << "(3) 查詢帳上餘額" << endl;
    cout << "(0) 結束" << endl;
    cout << "請選擇服務項目(0-3):";

    int i;
    cin >> i;

    // 以下用 switch ... case ... 改寫
    switch(i)
    {
        case 1:
            cout << "進行提款作業中..." << endl;
            cout << "提款作業完成!" << endl;
            break;
        case 2:
            cout << "進行存款作業中..." << endl;
            cout << "存款作業完成!" << endl;
            break;
        case 3:
            cout << "進行查詢作業中..." << endl;
            cout << "查詢作業完成!" << endl;
            break;
        case 0:
            cout << "結束服務" << endl;
            break;
        default:
            cout << "無此項目" << endl;
    }

    return 0;
}
```

break 很重要

使用 `switch ... case ...` 時最常發生的錯誤就是忘記加上 **break**。

case x: 只是一個標籤，程式跳到符合的標籤後開始向下執行，若沒遇到 **break** 會一直向下執行下去，即使遇到另一個 case 也是一樣。

```
    int num = 1;

    // 這不是我們想要的
    switch(num) {
        case 1:
            cout << "this number is 1." << endl;
        case 2:
            cout << "this number is 2." << endl;  // num 為 1 時這行也會執行到
    }
```

程式輸出如下:

```
this number is 1.
this number is 2.
```

```
    int num = 1;
    // 這才是我們想要的
    switch(num) {
        case 1:
            cout << "this number is 1." << endl;
            break;
        case 2:
            cout << "this number is 2." << endl;
            break;
    }
```

程式輸出如下:

```
this number is 1.
```

Fall-through - 利用沒加 break 的副作用

有時候我們會故意不加 **break** 利用它會一直執行下去的副作用來達到特別的目的。

練習：各月份所屬的季節

讀取使用者輸入的一個整數 $m, (1 \leq m \leq 12)$，輸出其所屬的季節。

- 2, 3, 4 月：春
- 5, 6, 7 月：夏
- 8, 9, 10 月：秋
- 11, 12, 1 月：冬

這樣寫看起來很累贅。

```
    int m;
    cin >> m

    switch(month) {
        case 2:
            cout << "Spring" << endl;
            break;
        case 3:
            cout << "Spring" << endl;
            break;
        case 4:
            cout << "Spring" << endl;
            break;
        case 5:
            cout << "Summer" << endl;
            break;
        case 6:
```

```
            cout << "Summer" << endl;
            break;
        case 7:
            cout << "Summer" << endl;
            break;
        case 8:
            cout << "Fall" << endl;
            break;
        case 9:
            cout << "Fall" << endl;
            break;
        case 10:
            cout << "Fall" << endl;
            break;
        default:
            cout << "Winter" << endl;
    }
```

這樣寫就好多了。

```
    int m;
    cin >> m

    switch(month) {
        case 2:
        case 3:
        case 4:
            cout << "Spring" << endl;
            break;
        case 5:
        case 6:
        case 7:
            cout << "Summer" << endl;
            break;
        case 8:
        case 9:
        case 10:
            cout << "Fall" << endl;
            break;
        default:
            cout << "Winter" << endl;
    }
```

3.5 三元運算子？：

3.5.1 三元運算子？：

在 C++ 中，三元運算子（Ternary Operator）是唯一一個需要三個運算元的運算子。它的符號是？和：。

這個運算子主要用來取代簡單的 if-else 判斷式，讓程式碼在一行內就能完成條件判斷與賦值，非常方便。

語法

三元運算子的基本語法結構如下：

```
條件式 ? 運算式1 : 運算式2;
```

條件式 (Condition): 這是一個會回傳 true (真) 或 false (假) 的布林表達式。

- **? :** 這是三元運算子的核心符號。
- **運算式1 (Expression1):** 如果「條件式」的結果為 true，則執行這個運算式，並將其結果作為整個三元運算式的最終結果。
- **運算式2 (Expression2):** 如果「條件式」的結果為 false，則執行這個運算式，並將其結果作為整個三元運算式的最終結果。

3.5.2 實例練習

讓我們來看幾個例子，比較一下使用 if-else 和使用三元運算子的差別。

範例：判斷奇偶數

假設我們要讓使用者輸入一個整數，然後判斷它是奇數還是偶數。

傳統的 if-else 寫法：

```
    int number;
    cout << "請輸入一個整數: ";
    cin >> number;

    string result;
    if (number % 2 == 0) {
        result = "偶數";
    } else {
        result = "奇數";
    }

    cout << "這個數字是 " << result << endl;
```

使用三元運算子的寫法：

```
    int number;
    cout << "請輸入一個整數: ";
    cin >> number;

    // 一行就搞定！
    string result = (number % 2 == 0) ? "偶數" : "奇數";

    cout << "這個數字是 " << result << endl;
```

解說：

在三元運算子的版本中，(number % 2 == 0) 是我們的條件式。

- 如果 number 除以 2 的餘數為 0 (條件為 true)，則回傳？後面的字串 "偶數"。
- 如果餘數不為 0 (條件為 false)，則回傳：後面的字串 "奇數"。

- 最後，回傳的字串會被直接賦值給 result 變數。是不是簡潔很多呢？

你還可以進一步像這樣直接使用他的運算結果

```
    int number;
    cout << "請輸入一個整數: ";
    cin >> number;

    // 注意：要用 ( ) 包覆整個運算式
    cout << "這個數字是 " << ((number % 2 == 0) ? "偶數" : "奇數") << endl;
```

範例：絕對值計算

```
    int a;
    cout << "請輸入一個整數 a: ";
    cin >> a;

    cout << "|a|= " << ((a < 0) ? -a : a) << endl;
```

3.5.2 使用時機與注意事項

- **優點：** 程式碼簡潔。
- **缺點：** 不適合處理複雜的邏輯。如果 `if` 或 `else` 區塊中需要執行多行程式碼，就不應該使用三元運算子，否則會讓程式碼變得難以閱讀和維護。
- **原則：** 當 `if-else` 只是為了根據一個簡單條件來賦予變數不同的值時，就是使用三元運算子的最佳時機。

三元運算子 `? :` 是一個非常實用的語法糖 (Syntactic Sugar)，能讓你用更精簡的方式寫出條件判斷。熟練使用它可以提升程式碼的美觀與效率，但切記不要濫用，以免降低複雜邏輯的可讀性。