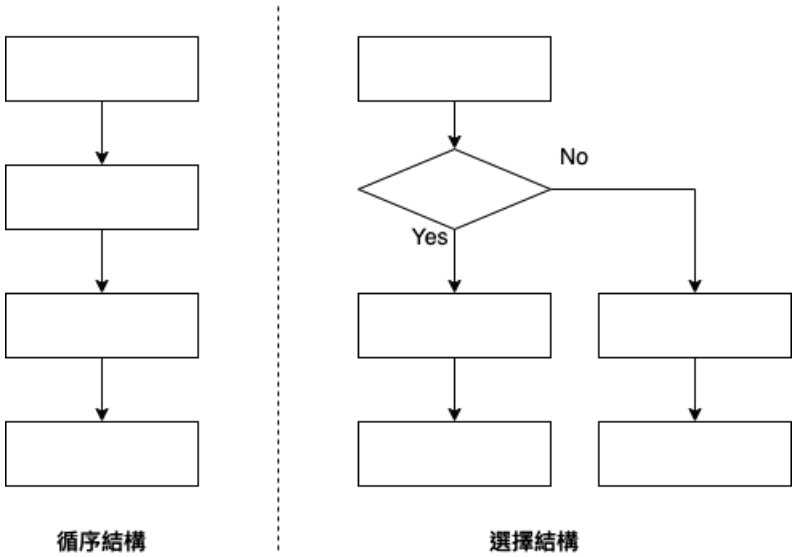
4.1 while 迴圈

程式執行流程結構

目前為止我們學過了兩種程式執行的流程結構，(1)循序結構；(2)選擇結構。



接下來我們要學的是 **重覆結構**，也就是可以重覆執行一段程式。

while

練習：輸出一行，共 5 個 '*'

這個很簡單，只要一行 cout 就能搞定。

```
cout << "*****" << endl;
```

那如果是這題呢？

練習：輸出一行，共 375 個 '*'

我們不太可能傻傻的在字串裡一邊打字一邊數 375 個吧？我們想要的是重覆 `cout << '*';` 375 次。而且要簡單明瞭，不是複製後貼上 375 次。

在這裡我們引入 `while` 敘述，它可以在指定條件成立時，不斷重覆指定的工作，直到該條件不再成立為止。

while 的基本語法如下：

```
while( 條件判斷式 )
{
    條件成立時要重覆做的事
}
```

就輸出 375 個 '*' 來說，使用 while 可以這麼做。

```
int i=1;

while(i<=375)
{
    cout << '*';
    i = i+1;  // 這行如果忘記，迴圈永遠不會結束
```

```
}
cout << endl;
```

請注意在這個 while 迴圈中，變數 i 擔任的角色。它一開始的初值是 1，每執行一次會遞增 1，當 i 大於 375 之後，就離開迴圈。這是一個「計數器」的角色，我們利用一個變數來記錄這個迴圈繞到第幾圈了。

練習：輸出一行，共 n 個 '*'

如果我們要程式更有彈性一點，由使用者指定要輸出的 '*' 數量 n。

只要把前一個程式的 375 改成變數 n 即可。

```
int n;
cin >> n;

int i=1;

while(i<=n)
{
    cout << '*';
    i = i+1;
}
cout << endl;
```

雖然在前面的例子裡我們說 i 擔任「計數器」的角色，但它本質上就只是個變數，也可以參與到迴圈內的計算、輸出……。

練習：輸出 1~n

在這個例子裡，我們在迴圈的每一圈輸出 i 當下的值。

```
int n;
cin >> n;

int i=1;
while(i<=n)
{
    cout << i << " ";
    i = i+1;
}
cout << endl;
```

```
5
1 2 3 4 5
```

接下來這題我們來看兩種做法。

輸出 1~n 之間的奇數

方法一： 首項為1，公差為2 的等差數列

```
int n;
cin >> n;

int i=1;  // 首項為 1
while(i<=n)
{
    cout << i << " ";
    i = i+2;  // 公差為 2
}
cout << endl;
```

```
10
1 3 5 7 9
```

方法二： 在 1~n 之間，逐一過濾符合條件的才輸出

```
int n;
cin >> n;

int i=1;
while(i<=n)
{
    if(i%2==1)
    {
        cout << i << " ";
    }
    i = i+1;
}
cout << endl;
```

```
10
1 3 5 7 9
```

就上題來說，**方法一** 比較有效率。但有時候除了逐一過濾檢查之外，沒有更好的辦法。

練習：輸出 n 的所有正因數

某個整數的因數，可不會簡單到成等差數列分布。n 的所有正因數是在 1 ~ n 之間每個可以整除 n 的整數。

```
int n;
cin >> n;
cout << n << " 的正因數有：";

int i=1;
while(i<=n)
{
    if(n%i==0)
    {
        cout << i << " ";
    }
    i = i+1;
}
cout << endl;
```

```
16
16 的正因數有：1 2 4 8 16
```

練習：n 有幾個正因數？

這個例子裡，我們要的是正因數的數量，作法為：

1. 將一個用來計數用的變數歸零
2. 每發現一個正因數，就將該計數累加 1
3. 最後輸出該計數值

```
int n;
cin >> n;

int sum = 0; // 一開始要給定初值歸零

int i=1;
while(i<=n)
{
    if(n%i==0)
    {
        sum = sum+1;
    }
    i = i+1;
}
cout << n << " 有 " << sum << " 個正因數" << endl;
```

```
16
16 有 5 個正因數
```

break - 跳出迴圈

有時候我們使用迴圈的目的是要在一個可能範圍中 (1)找出特定目標，(2)確定某條件。所以一但找到或確定了，就可以離開迴圈，不必繼續把後面的圈數跑完。

質數判定

在數學和電腦科學中，判定一個正整數是否為質數是很重要的。如果你去 google 一下，會發現方法有非常多種。在這裡我們使用一個最簡單的概念來實作這個判定 - **「如果一個正整數 n 只有 1 和 n 這兩個因數，則 n 為質數」**。

練習：質數判定(1)

使用前一個「n 有幾個正因數？」程式碼，可以很快完成這個質數判定程式。

正因數數量為 2 的是質數。其他的都不是質數。

```
int n;
cin >> n;

int sum = 0;

int i=1;
while(i<=n)
{
    if(n%i==0)
    {
        sum = sum+1;
    }
    i = i+1;
}

if(sum==2)
{
    cout << n << " 是質數" << endl;
}
else
{
    cout << n << " 不是質數" << endl;
}
```

練習：質數判定(2)

我們也可以這權想：在 2 ~ (n-1) 之間，如果發現任一個 n 的因數，那麼 n 就不是質數，反之 n 為質數。

下面這個例子，我們先假設 n 是質數，記錄在布林型別的變數 is Prime 中(isPrime = true)。

接著嘗試在 2 ~ (n-1) 之間試試看能否找到 n 的因數。若找到了，表示 n 不是質數，把這個事實記錄下來(isPrime = false)。

在把所有可能的因數都看完後，由 isPrime 的值即可判定 n 是否為質數。

```
int n;
cin >> n;

bool isPrime = true; // 先假設 n 是質數 (n is a prime number)

int i=2;
while(i<n)
{
    if(n%i==0)
```

```
    {
        isPrime = false;  // 如果發現任一 n 的因數，修正 isPrime 為 false
    }
    i = i+1;
}

if(isPrime)
{
    cout << n << " 是質數" << endl;
}
else
{
    cout << n << " 不是質數" << endl;
}
```

練習：質數判定(3)

上面這個判定質數演算法是可行的，但是效率上有頗多浪費之處。例如 n=256，明明一開始我們就發現 2 是 n 的因數，當下就可以判定 n 不是質數，但卻還是把剩下的 253 圈(i=3~255)跑完。

在第 11 行之後，我們就可以跳出迴圈了。

在這裡我們可以使用 `break` 敘述，程式執行到 break 時，跳出當下所在的那一層迴圈。

雖然只是加了這一行，卻可以省下大量的時間。

```
int n;
cin >> n;

bool isPrime = true; // 先假設 n 是質數 (n is a prime number)

int i=2;
while(i<n)
{
    if(n%i==0)
    {
        isPrime = false;
        break;  // 跳出迴圈
    }
    i = i+1;
}

if(isPrime)
{
    cout << n << " 是質數" << endl;
}
else
{
    cout << n << " 不是質數" << endl;
}
```

continue - 繼續下一圈

想像一下這個場景，你是一個在櫃枱負責審核資料的員工，客戶按抽到號碼牌的順序來到你面前。對每個客戶，你要審查他給你的 20 張表單是否符合申請需求。

整個流程應該是像這樣。

```
num = 0

while(還沒到下班時間)
{
    num = num+1
    廣播請 num 號到櫃枱
    審查 表單1
    審查 表單2
    審查 表單3
```

```
    ......
    審查 表單20
}
```

如果今天有個客戶，他的表單 3 不符資格，當下你就可以告知他審查結果為「不符資格」，並請下一位客戶過來櫃枱，無需再把他後面的 17 張表格看完。

`continue` 就是這樣一個「下一位」的敘述。程式執行到 continue 時，會略過當下那圈剩下的所有工作，直接回到迴圈的開頭並繼續執行下去。

練習：排除 1 ~ n 間，3 的倍數和尾數為 3 的數。

```
int n;
cin >> n;

int i=0;

while(i<n)
{
    i = i+1;
    if(i%3==0 || i%10==3)
    {
        continue;
    }
    cout << i << " ";
}
cout << endl;
```

```
16
1 2 4 5 7 8 10 11 14 16
```

注意！在 while 迴圈中，continue 只是立刻回到迴圈開頭處，判斷若條件成立便再進入迴圈執行。並不會自己幫你把 i 的值加 1。

所以若把上面的程式改成這樣是不會正確運作的。

```
int n;
cin >> n;

int i=1;

while(i<=n)
{
    if(i%3==0 || i%10==3)
    {
        continue;  // i 沒有遞增，會造成無窮迴圈
    }
    cout << i << " ";
    i = i+1;
}
cout << endl;
```

讀取若干組資料

在競技程式設計比賽時，很常見一種輸入資料不確定有幾組的狀況，例如以下這個例子。

練習：加總計算

輸入說明： 輸入為若干個整數

輸出說明： 請輸出這些整數的總合。

若干個？你根本不知道有幾個數要怎麼做？要讀到第幾個才能輸出？

在競賽中並不是由裁判手動在那裡用鍵盤輸入資料，他們早把要輸入的資料都寫到一個檔案裡，然後再把那個檔案 **餵給**

你的程式。

有時候也會註明，輸入資料以 EOF 做為結束，EOF 即 `End Of File`，就是檔案的結束。所以你要做的就是一直讀到沒有資料可以讀為止。

以下為常見的模版，使用 `while(cin>>a)` 迴圈來讀取不定數量的資料。

```
#include <iostream>

using namespace std;

int main()
{
    int sum = 0;
    int a;

    while(cin>>a)  // 順利讀到資料即為 true，否則為 false
    {
        sum = sum+a;
    }
    cout << sum << endl;

    return 0;
}
```

Revision #31
★Created 12 March 2024 02:10:24 by huihui
Updated 25 March 2024 12:51:03 by huihui