國立屏東大學 資訊工程學系 C++程式設計入門教材

8. 迴圈

Chicago的捷運系統自1892年起開始營運(不過它還不是世界上最早的捷運系統，London的捷運自1863年起就開始營運了)，目前共有八條路線，大部份路線為高架或地面路線，Chicago的市中心被稱為路普區，這是因為當捷運從四面八方進入市區後，會以環狀的方式繞行市中心一圈後離開；從早到晚，這種不斷有捷運繞圈運行的路線，讓市中心漸漸地被人稱為Loop區(中譯為路普區) — 意思就是一直在繞圈圈的地方。

迴圈(Loop)也是程式設計三種基本組成結構之一，可以讓程式碼依特定的條件重複地執行。一個迴圈通常使用一組大括號將一些程式敘述包裹起來，並且可以重複地執行，直到特定的條件成立或不成立為止。這些被包裹起來被重複執行的程式碼被稱為迴圈主體(Loop Body)，其用以判斷迴圈是否要繼續執行的條件，則稱為測試條件(Test Condition)，通常是一個運算結果必須為布林值(true或false)的邏輯運算式(Logical Expression)，至於判斷是否繼續執行的地方，可以在迴圈區塊的進入點(Entry Point，意即開頭處)或是離開點(Exit Point，意即結束處)，視所使用的迴圈敘述而定。

C++語言支援三種迴圈敘述，同樣都可以讓特定的程式碼重複執行，只是其進入點、離開點與或測試條件的位置與語法不同而已。本章將先從while迴圈敘述開始介紹C++語言所支援的迴圈敘述，後續再針對do while與for迴圈敘述加以說明。

8.1 while迴圈

8.1.1 語法

while迴圈敘述可讓特定的程式碼反覆執行，直到特定條件不成立為止，其語法如下：

while迴圈敘述語法

while (測試條件) 敘述 | { 敘述* }

while敘述先判斷「測試條件」的值，若為true則會執行後續的一行敘述，或是由一組大括號所包裹起來的多行敘述，直到「測試條件」的值為false時才結束。我們將while迴圈所要重複執行的一行或多行敘述，稱為其「迴圈主體(Loop Body)」。在while迴圈執行時，依據「測試條件」的布林結果，其迴圈主體可能一次都不執行(第一次進入迴圈時，其測試條件就為false)，或是可無限次數的執行下去(每次測試都為true)。請參考figure 1，它將while迴圈執行時的過程以流程圖加以表達。

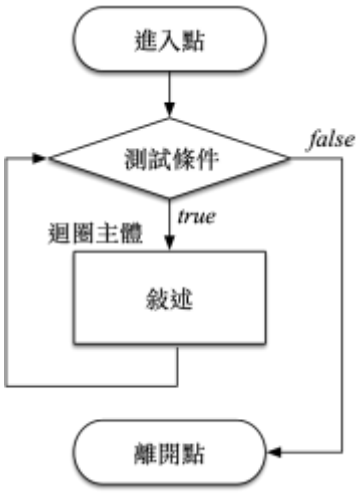


Fig. 1: while迴圈運作流程

以下三個例子都是讓while迴圈重複執行多次，直到變數i的數值不再大於0為止，只不過其迴圈主體所包含的程式敘述不盡相同而已：

```
int i=5;
while(i>0)       //迴圈主體只有一行敘述
   cout << i--; //輸出i的數值後將它遞減1
```

```
int i=5;
while(i>0)        // 迴圈主體使用一組大括號包裹起來，但裡面仍只有一行敘述
{
   cout << i--;  // 輸出i的數值後將它遞減1
}
```

```
int i=5;
while(i>0)       // 迴圈主體使用一組大括號包裹起來
{
   cout << i;    // 輸出i的數值後將它遞減1
   i--;  // 輸出i的數值後將它遞減1
}
```

上述的三個程式，它們的執行結果都是相同的：

```
54321
```

8.1.2 應用範例

以下是一些while迴圈的應用範例：

```
// 計算1+2+3+...+10的結果
int i=1, sum=0;

//執行十次
while(i<=10)
{
   sum+=i;
   i++;
}
cout << "sum of 1 to 10 is " << sum << endl;
```

```
// 印出介於1到100間可以被7整除的數字
int i=1;

while(i<=100)
{
   if(i%7==0)
      cout << i << endl;
   i++;
}
```

```
// 反覆執行直到使用者輸入'q'為止
bool quit=false; //宣告quit變數，其初始值false表示"沒有要"離開程式的執行
char c;

while(!quit)
{
   // do something
   ...
   cout << "Continue?(y/n)";
   cin >> c;
   if(c=='n')
      quit=true;
}
```

8.1.3 無窮迴圈(infinite loop)

不正確的使用迴圈有可能會發生測試條件永遠成立(意即永遠都為true)的情況，其結果將會使得迴圈永遠不會結束其執行 — 我們將此種情況稱為無窮迴圈(Infinite Loop)。以下幾個例子，是在迴圈主體裡沒有能夠改變「測試條件」的程式碼，使得迴圈永無止境地不斷執行：

```
int i=5;
```

```
while(i>0)       // 迴圈主體使用一組大括號包裹起來
{
   cout << i;    // 輸出i的數值後將它遞減1
      // 忘了寫i--去改變其數值，因此其測試條件i>0將永遠成立
}
```

```
int i=5;
while(i>0)       // 迴圈主體使用一組大括號包裹起來
{
   cout << i;    // 輸出i的數值後將它遞減1
   i++;          // 把遞減錯寫為遞增，因此其測試條件i>0將永遠成立
}
```

```
// 反覆執行直到使用者輸入'q'為止
bool quit=false;
char c;
int count=0;

while(quit=false)  //只要quit的值為false就繼續執行，但此處不小心將quit==false寫成了quit=false
{
   // do something
   ...
   if(expression)   //若特定條件成立則將quit變數設為true，表示要離開程式的執行
      quit=true;
}
```



注意：發生無窮迴圈該怎麼讓程式停止執行？

使用Ctrl+C將程式跳離，Mac OS系統請使用Control+C，然後在Linux/Mac OS系統可以使用ps aux指令查看程式的PID，再以kill -9 PID指令將程式從系統中移除。至於Windows系統，則可以使用tasklist指令查看程式的PID，再以taskkill /PID -t指令將程式從系統移除。

8.2 do while迴圈

do while迴圈敘述和while迴圈是類似的，都適用於在特定條件滿足以前，不斷重複執行迴圈主體的一種結構；但不同於while迴圈在進入點(開始執行迴圈時)進行「測試條件」的判讀，do while迴圈是在每次完成迴圈主體的執行後才進行測試條件的判讀 — 若判斷結果為true則繼續回到do while迴圈開頭處再次執行；相反地，若測試條件的結果為false時，do while迴圈就會結束。

8.2.1 語法

do while迴圈的語法如下：

do while迴圈敘述語法

do 敘述 | { 敘述* } while(測試條件);

相較於while迴圈，由於do while迴圈測試條件是在迴圈主體結束後才加以檢查，所以其迴圈主體內容至少會被執行一次；反之□while迴圈在測試條件不成立的情況下，其迴圈主體有可能一次都沒有執行。請參考figure 2，它將do while迴圈的運作過程以流程圖加以表達。

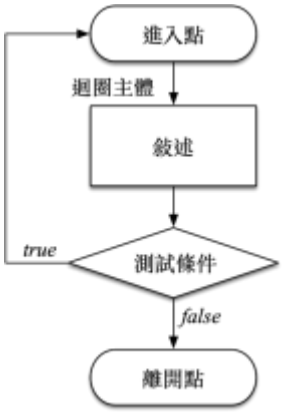


Fig. 2: do while迴圈運作流程

別忘了在do while迴圈後的分號

與while迴圈不同的是，依語法do while迴圈最後面必須加上一個分號做為結尾。但可能是受到while迴圈的影響，很多人在寫程式時，都忘了要為do while迴圈加上分號。

8.2.2 應用範例

以下是一些例子：

```
//輸出10..9..8..7..6..5..4..3..2..1
int i=10;

do
{
     cout << i << ".." ;
```

```
     i--;
} while (i>0);
cout << endl;
```

```
//輸出10..9..8..7..6..5..4..3..2..1
int i=10;

do
{
     cout << i << "..";
} while (--i>0);
cout << endl;
```

```
// 讓使用者重複輸入一個代表分數的整數，直到其值介於0~100為止
int score;
do
{
    cout << "Please input a score (between 0 to 100): ";
    cin >> score;
} while( (score<0) || (score>100) );
...
```

這個程式片段在許多應用中都可以看到，其作用是限制使用者只能輸入特定範圍的數值，其執行結果可參考如下：

```
Please input a score (between 0 to 100): -5⏎
Please input a score (between 0 to 100): 111⏎
Please input a score (between 0 to 100): 66⏎
```

下面這個程式會要反覆地要求使用者輸入兩個整數a與b□直a可以被b整除為止:

```
int a, b;
do
{
    cout << "Please input two integers: ";
    cin >> a >> b;
} while((a%b)!=0);
```

其執行結果如下：

```
Please input two integers: 3 5⏎
Please input two integers: 13 5⏎
Please input two integers: 13 15⏎
```

```
Please input two integers: 23 5⏎
Please input two integers: 400 20⏎
```

8.3 for迴圈

for迴圈是C++語言所支援的第三種迴圈結構，但它與前兩者(也就是while與do while迴圈)比較不同，通常for迴圈的執行必須搭配一個用以控制迴圈執行次數的迴圈變數(Loop Variable，亦稱為迭代變數Iteration Variable)，在運行時先使用初始化敘述(Initialization Statement)對迴圈變數進行初始化的動作，然後開始進行迴圈的測試條件(Test Condition)判斷(通常此測試條件也與迴圈變數相關)，若結果為true則進入迴圈主體(Loop Body)執行，若結果為false則結束迴圈；每次迴圈主體執行完後，還必須使用更新敘述(Update Statement)對迴圈變數執行更新的動作，請參考figure 3的流程圖。

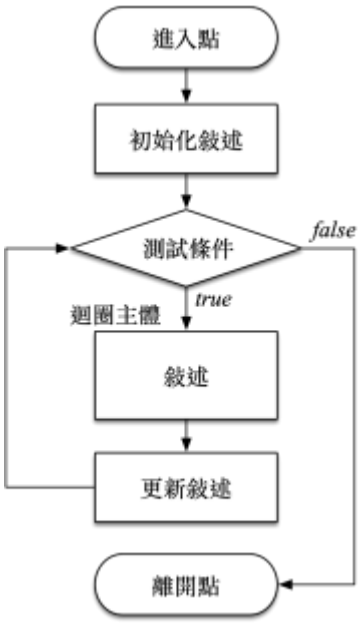


Fig. 3: for迴圈運作流程

8.3.1 語法

for敘述的語法如下：

for迴圈敘述語法

for (初始化敘述; 測試條件; 更新敘述) 敘述 | { 敘述* }

其中初始化敘述、測試條件與更新敘述，分別是用以定義迴圈的初始條件、中止條件與更新的處理；要注意的是，其初始條件、中止條件與更新通常都是針對迴圈變數所設計。以下分別加以說明：

- 初始化敘述：在迴圈初次執行前被執行，通常用以設定迴圈變數的初始值。

- 測試條件：在迴圈每次執行前加以檢查，視其值決定是否繼續執行；若其值為true則繼續，反之若其值為false則結束。此測試條件為一個邏輯運算式，其中通常包含有迴圈變數做為其運算元之一。
- 更新敘述：在迴圈主體每次被執行完後加以執行，通常是用來更新迴圈變數的值。

for迴圈可以轉換為等價的while迴圈

基本上for迴圈與while迴圈是可以互相轉換的，例如我們可以使用while的語法來將for的語法加以改寫：



```
初始化敘述;
while(測試條件)
{
   敘述 | { 敘述* }
   更新敘述;
}
```

8.3.2 應用範例

以下是一些應用範例：

```
int i,sum=0;

for(i=1;i<=10;i++)
{
   sum+=i;
}
cout << "sum=" << sum << endl;
```

我們也可以在初始化敘述與更新敘述裡，使用逗號運算子','用來同時指定多個運算式，例如:

```
int i,sum;

for(i=1, sum=0;i<=10;i++)
{
   sum+=i;
}
cout << "sum=" << sum << endl;
```

甚至初始化敘述與更新敘述也可以被省略，例如:

```
int i=0;

for( ; i<10;i++)
   cout << "i=" << i << endl;
```

8.4 巢狀迴圈

一個迴圈內如含有另一個迴圈，則稱之為，巢狀迴圈(Nested Loop)。每一層的迴圈可以是for□while或do while其中任一個，以下我們僅以for迴圈為例，其它的組合您可以自行代換。

請參考以下的範例，它使用一個迴圈讓變數i從1執行到10，再用一個內層的迴圈計算i的階乘並把計算出來的階乘值加總：

```
//印出1!+2!+3! + ... + 10!
int i,j,temp,sum=0;

for(i=1;i<=10;i++)
{
   temp=1;
   for(j=1; j<=i; j++)
   {
      temp*=j;
   }
   sum += temp;
}

cout << "sum=" << sum << endl;
```

請思考以下問題：

- 第6行的temp=1可不可以省略？
- 可以把第6行併入第4行，寫做□for(i=1, temp=0; i<=10; i++□嗎？
- 同理，第2行的sum=0也可以併入嗎？

其實要計算1到10的階乘的和並不一定要使用雙層的巢狀迴圈。下面這個範例僅使用了一個迴圈，就完成了1到10的階乘和之計算：

```
//印出1!+2!+3! + ... + 10!
int i,j,temp=1,sum=0;

for(i=1;i<=10;i++)
{
   temp*=i;
   sum += temp;
```

```
}
cout << "sum=" << sum << endl;
```

動動腦，自己試著把這個程式看懂吧！

8.5 從迴圈中跳離

除了使用迴圈的測試條件來控制迴圈的執行外，我們還可以使用break、continue與goto敘述來改變程式的動線，使其可以跳離迴圈所屬的程式區塊。

8.5.1 break敘述

我們可以在迴圈主體裡使用break敘述來跳離迴圈。在迴圈主體中的break敘述一旦被執行，則在此次迴圈執行過程中剩餘還未執行的敘述就會被跳過不執行，並且結束迴圈的執行。當迴圈的中止條件不在開頭或結尾時，break敘述就便得很有用處，例如：

```
//反覆要求使用者輸入一個整數，並且將其累加，直到使用者輸入0為止

int n, sum=0;

for(;;)
{
   cout << "Please input a number (0 for quit):";
   cin >> n;
   if(n==0)
      break;
   sum+=n;
}
cout << "sum=" << sum << endl;
```

再看看另一個範例：

```


while(true) // 測試條件直接寫成布林值true，所以此迴圈會不斷地執行
{
   // do something
   ...
   if(expression) //直到特定條件成立時，使用break跳離
      break;
}
```

8.5.2 continue敘述

continue則和break相反，它並不會結束迴圈的執行，而是省略當次執行時未完成的程式碼，直接執行迴圈的下一回合。

```
//反覆要求使用者輸入一個整數，並且將其累加，直到使用者輸入0為止，但輸入值若為負數則加以忽略
int n, sum=0;

for(;;)
{
   cout << "Please input a number (0 for quit):";
   cin >> n;
   if(n==0)
      break;
   if(n<0)
      continue;
   sum+=n;
   // continue敘述使程式碼跳到了這裡
}
cout << "sum=" << sum << endl;
```

8.5.3 goto敘述

C++語言還提供另一種無條件的跳躍敘述－goto敘述。我們可以在程式碼中的特定位置標記一些標籤(Label)，其方法為在某行以標記名稱後接冒號的方式來定義，爾後需要改變程式碼執行動線時，則使用goto 標記名稱;的方式即可完成。請參考以下的範例：

```
//反覆要求使用者輸入一個整數，並且將其累加，直到使用者輸入0為止
   int n, sum=0;

   for(;;)
   {
      cout << "Please input a number (0 for quit):";
      cint >> n;
      if(n==0)
         goto done;
      sum+=n;
   }

done:
   cout << "sum=" << sum << endl;
```

goto敘述不一定要配合迴圈的使用，例如:

```
int main()
{
char cmd;

begin:

   cin >> cmd;

   if(cmd != 'q')
      goto begin;
   cout << "Exit" << endl;
}
```

這個程式讓使用者不斷地輸入一個字元，直到其輸入字元為'q'時才結束程式。其中定義了一個名為begin的標記，在後續的if敘述裡，若其測試條件成立則使用goto敘述跳躍到begin標記處。



許多程式設計師一直在爭論是否該在程式碼中使用goto□正反兩面的意見都值得參考。我覺得如果您覺得好用就用吧？只是每次使用時也順便想一想：同樣的功能如果不使用goto可以做到嗎？以免以後你不用goto就不會寫程式！我所認識的程式設計師裡面，兩種人都有，不過反對使用goto的人，通常完全容不下在程式中使用goto□如果你擔心以後工作上的主管或同事不喜歡你寫的含有goto的程式，那你最好用與不用都會寫，這樣就沒問題了！

From:
https://junwu.nptu.edu.tw/dokuwiki/ - **Jun Wu的教學網頁**
國立屏東大學資訊工程學系
CSIE, NPTU
Total: 333456

Permanent link:
https://junwu.nptu.edu.tw/dokuwiki/doku.php?id=cppbook:ch-loop

Last update: **2024/01/12 07:38**