由片語學習C程式設計

台灣大學資訊工程系劉邦鋒著

台灣大學劉邦鋒老師講授

August 19, 2016

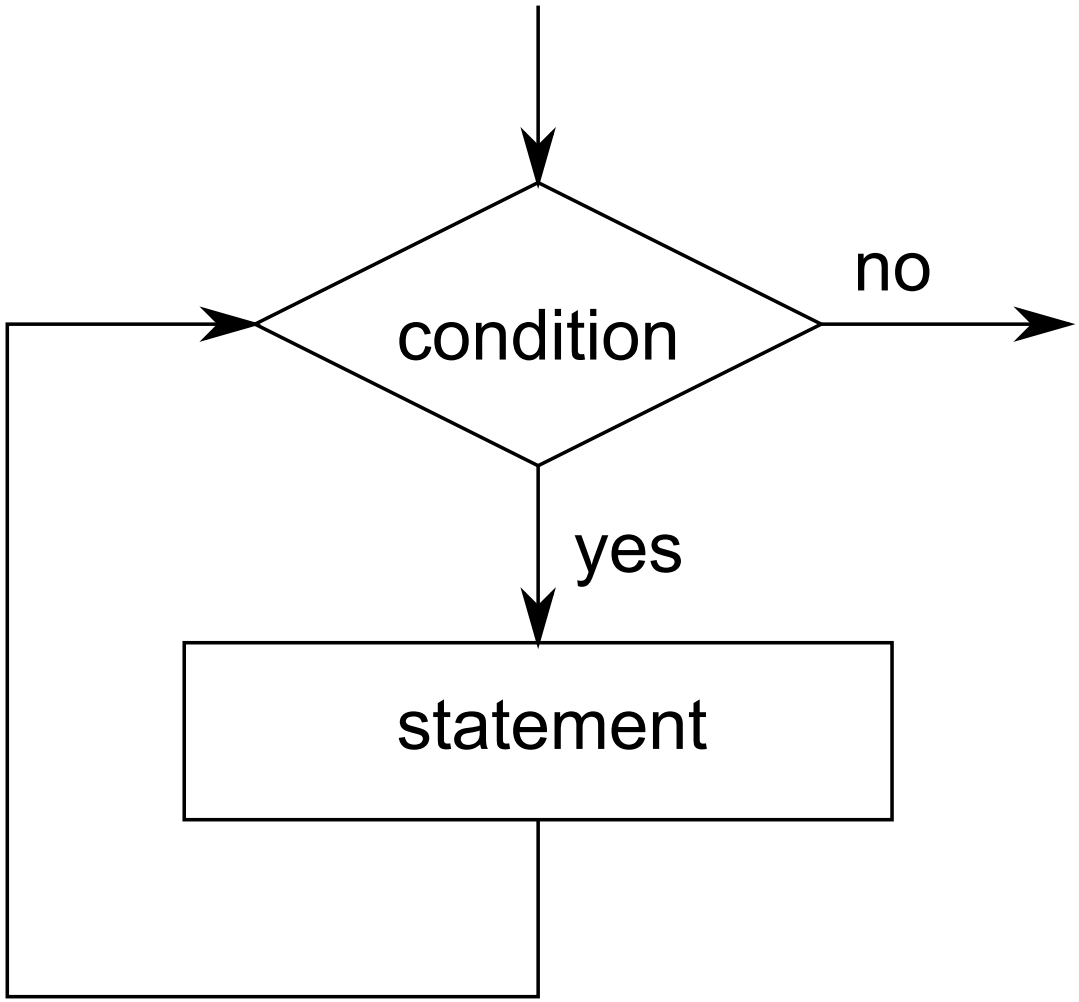
第五單元

迴圈

片語 1: while 迴圈

```
1 while (condition)
2   statement;
```

- 當條件為 **真** 時，會執行敘述，然後再回頭檢查條件並繼續下去。
- 條件必定會和敘述有關連。當不希望繼續做敘述時， 敘述必須將條件設為 **偽**，使 while 迴圈結束。


condition
no
yes
statement

範例程式 2: (not-less-than.c) 5 的倍數中不小於 j 的最小值

```
1  #include <stdio.h>
2  main()
3  {
4     int i = 0;
5     int j;
6     scanf("%d", &j);
7     while (i < j)
8        i += 5;
9     printf("%d\n", i);
10 }
```

輸入

```
1 23
```

輸出

```
1 25
```

片語 3: 使用複合敘述的 while 迴圈

```
1 while (condition) {
2   statement1;
3   statement2;
4   statement3;
5 }
```

範例程式 4: (sum.c) 計算 1 加到 k 的和

```
1  #include <stdio.h>
2  main()
3  {
4     int i = 1;
5     int sum = 0;
6     int k;
7     scanf("%d", &k);
8     while (i <= k) {
9        sum += i;
10       i++;
11    }
12    printf("%d\n", sum);
13 }
```

輸入

```
1 100
```

輸出

```
1 5050
```

計算最大公因數

$$gcd(i,j) = \begin{cases} j & \text{if } i \bmod j = 0 \\ gcd(j, i \bmod j) & \text{otherwise} \end{cases} \quad (1)$$

範例程式 5: (gcd.c) 計算最大公因數

```
1  #include <stdio.h>
2  main()
3  {
4     int i, j, k;
5     scanf("%d", &i);
6     scanf("%d", &j);
7     while (i % j != 0) {
8        k = i % j;
9        i = j;
10       j = k;
11    }
12    printf("%d\n", j);
13 }
```

輸入

```
1 72
2 56
```

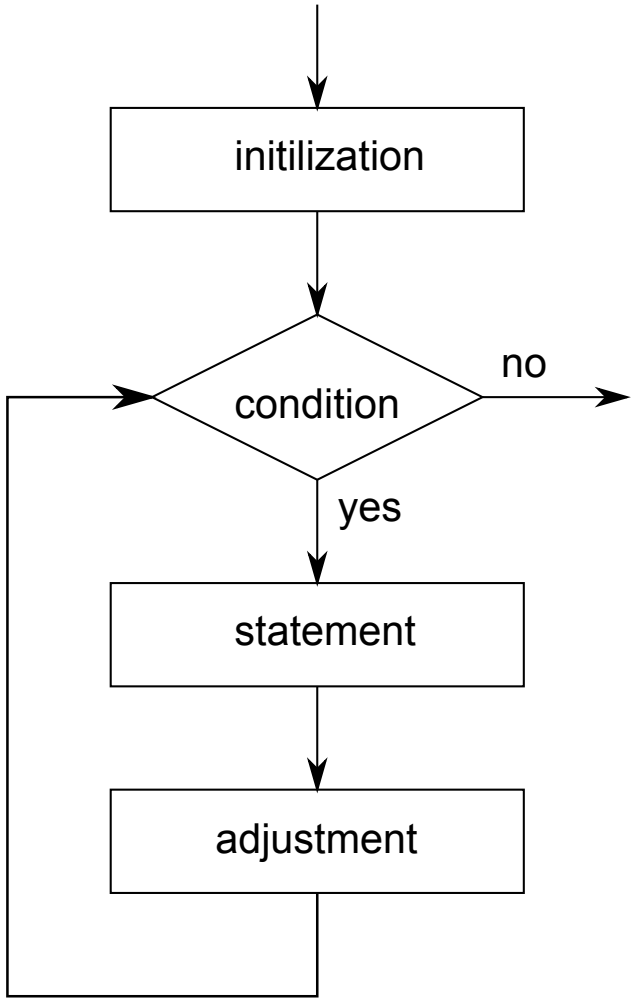
輸出

```
1 8
```

片語 6: for 迴圈

```
1 for (initialization; condition; adjustment)
2   statement;
```

- 首先執行初始化 (initialization) 的部分。
- 每一次進迴圈之前檢查條件。
- 若條件為 **真** 則執行敘述， 再執行調整 (adjustment) 並回到迴圈的開始。
- 若條件為 **偽**，則結束迴圈。


initilization
condition
no
yes
statement
adjustment

片語 7: 以 while 迴圈來表示 for 迴圈

```
1 initialization;
2 while (condition) {
3   statement;
4   adjustment;
5 }
```

學習要點

for 迴圈的初始化只會執行一次。

學習要點

for 迴圈先檢查條件，若為真才會執行敘述，所以有可能完全不執行敘述.

特殊字元

`for` 迴圈用分號 `;` 隔開初始化、條件、以及調整。

片語 8: 固定次數的 for 迴圈

```
1 for (i = 0; i < 100; i++)
2   statement;
```

- i 的值在初始化中被設為 0。
- 每一次進迴圈前檢查條件 (i < 100)。
- 若條件為 **真**，則執行敘述， 再執行調整 (i++)，並回到迴圈的開始。若條件為 **偽**，則結束迴圈。

範例程式 9: (for-print.c) 在 for 迴圈中列印 i 的值

```
1 #include <stdio.h>
2 main()
3 {
4   int i;
5   int j;
6   scanf("%d", &j);
7   for (i = 0; i < j; i++)
8     printf("%d\n", i);
9 }
```

輸入

```
1  10
```

輸出

```
1  0
2  1
3  2
4  3
5  4
6  5
7  6
8  7
9  8
10 9
```

質數判斷

- 判斷輸入的正整數 n 是否為質數。
- 試著用 2 到 $\sqrt{n}$ 去除 n，如果餘數為 0， 則設定 j 為 1。如果旗標 j 始終為 0，則 n 為質數。
- j 是做為一個 **旗標**，我們用它來判斷一件事是否發生過。

學習要點

使用旗標可判斷一件事是否發生過，但須注意旗標的初始值及調整方法。

範例程式 10: (prime.c) 判斷 n 是否為質數

```
1  #include <stdio.h>
2  main()
3  {
4     int i;
5     int j = 0;
6     int n;
7     scanf("%d", &n);
8     for (i = 2; (i * i) <= n; i++)
9        if ((n % i) == 0)
10          j = 1;
11    printf("%d\n", j);
12 }
```

輸入

```
1 73
```

輸出

```
1 0
```

片語 11: 使用複合敘述的固定次數 for 迴圈

```
1 for (i = 0; i < 100; i++) {
2   statement1;
3   statement2;
4   statement3;
5 }
```

範例程式 12: (sum-square-cubic.c) 計算和、 平方和、 及立方和

```c
#include <stdio.h>
main()
{
  int i, n;
  int sum = 0;
  int square_sum = 0;
  int cubic_sum = 0;
  scanf("%d", &n);
  for (i = 1; i <= n; i++) {
    sum += i;
    square_sum += (i * i);
    cubic_sum += (i * i * i);
  }
  printf("%d\n", sum);
  printf("%d\n", square_sum);
  printf("%d\n", cubic_sum);
}
```

輸入

```
1 100
```

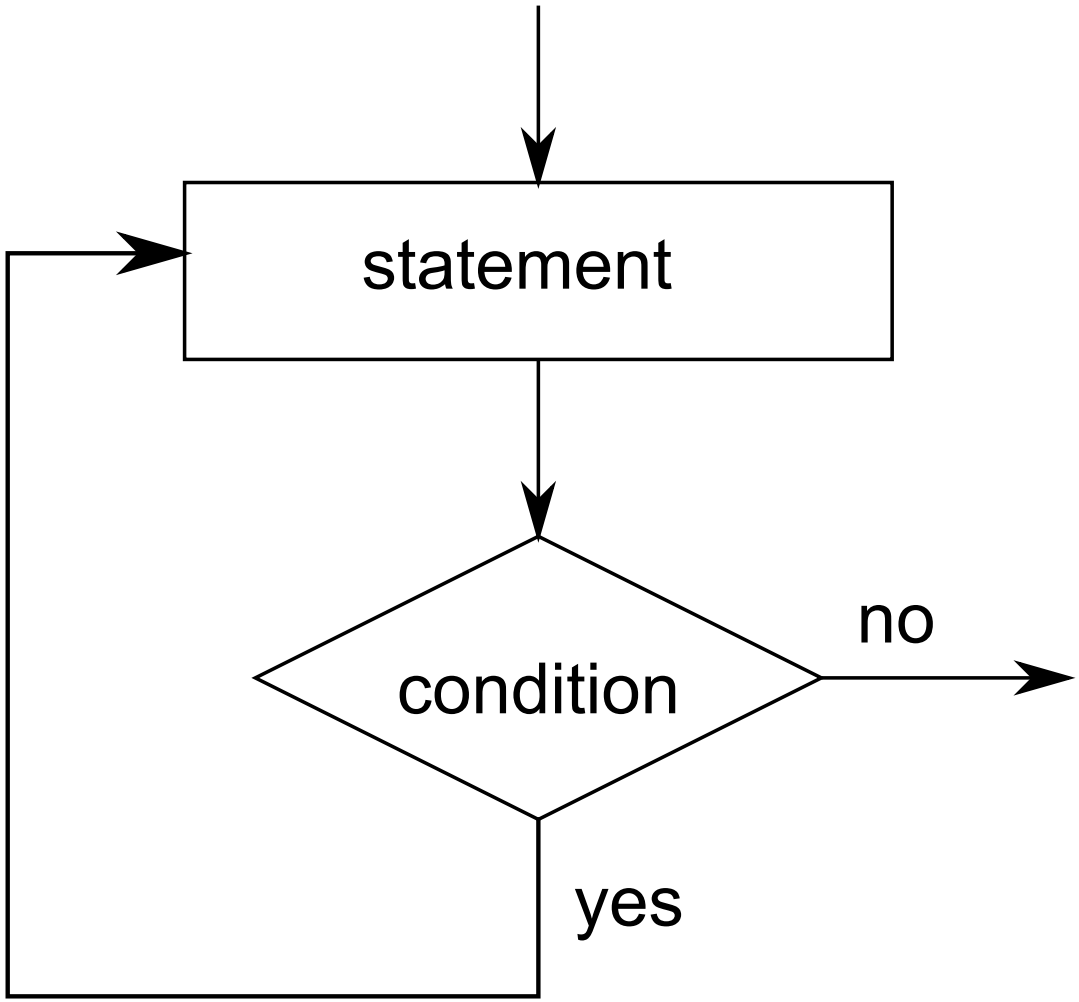
輸出

```
1 5050
2 338350
3 25502500
```

片語 13: do while 迴圈

```
1 do
2    statement;
3 while (condition);
```

- 先執行迴圈的主體部分一次，再測試條件。
- 如果條件成立，則繼續執行迴圈的主體部分。
- 如果條件不成立，則跳出迴圈。


statement
condition
no
yes

學習要點

`do while` 迴圈的主體部分至少會執行一次。

範例程式 14: (do-while.c) 將 i 由 100 加到 101

```
1  #include <stdio.h>
2  main()
3  {
4     int i;
5     scanf("%d", &i);
6     do
7        i++;
8     while (i < 0);
9     printf("%d\n", i);
10 }
```

輸入

```
1 100
```

輸出

```
1 101
```

片語 15: 使用複合敘述的 do while 迴圈

```
1 do {
2    statement1;
3    statement2;
4    statement3;
5 } while (cond);
```

範例程式 16: (sum-square-cubic-do-while.c) 計算和，平方和，及立方和

```
4   int i = 1;
5   int sum = 0;
6   int square_sum = 0;
7   int cubic_sum = 0;
8   int n;
9   scanf("%d", &n);
10  do {
11    sum += i;
12    square_sum += (i * i);
13    cubic_sum += (i * i * i);
14    i++;
15  } while (i <= n);
16  printf("%d\n", sum);
17  printf("%d\n", square_sum);
18  printf("%d\n", cubic_sum);
```

輸入

```
1 100
```

輸出

```
1 5050
2 338350
3 25502500
```

break

- 有時為了程式執行效能或是流程需要，我們希望提早結束一個迴圈。
- 當程式執行到 `break` 時，就會跳出迴圈，直接執行迴圈之後的敘述。
- 通常使用 `if then` 來判斷我們是否需要跳出迴圈。

片語 17: 用 break 提早結束一個 while 迴圈

```
1 while (loop_condition) {
2   ...
3   if (break_condition)
4     break;
5   ...
6 }
```

片語 **18:** 用 break 提早結束一個 for 迴圈

```
1 for (initialization; loop_condition; adjustment) {
2   ...
3   if (break_condition)
4     break;
5   ...
6 }
```

片語 19: 用 break 提早結束一個 do while 迴圈

```
1 do {
2    ...
3    if (break_condition)
4       break;
5    ...
6 }
7 while (loop_condition);
```

範例程式 20: (prime-break.c) 用 break 提早結束 for 迴圈。

```
1  #include <stdio.h>
2  main()
3  {
4    int i, j = 0, n;
5    scanf("%d", &n);
6    for (i = 2; (i * i) <= n; i++)
7      if ((n % i) == 0) {
8        j = 1;
9        break;
10     }
11   printf("%d\n", j);
12 }
```

- 當發現 n 有一個因數 i 時，不必再浪費時間去測試之後的 i。
- 這時我們就可以將 j 旗標設為 1 後，直接用 break 跳出 for 迴圈。

輸入

```
1 100
```

輸出

```
1 1
```

結束迴圈

- 使用 `break` 提早結束一個迴圈會讓程式的流程較為模糊，因為我們希望一個迴圈只有一個入口及一個出口。
- 如果迴圈可以因為不滿足迴圈條件而結束，或是由 `break` 結束， 則就會有兩個出口，這樣會讓讀程式的人不容易了解程式的流程。
- 改進的方法是使用旗標將 `break` 的條件也納入迴圈繼續執行的條件， 這樣就可以避免兩個出口的問題。

範例程式 21: (prime-flag.c) 用旗標提早結束 for 迴圈。

```
1  #include <stdio.h>
2  main()
3  {
4     int i;
5     int j = 0;
6     int n;
7     scanf("%d", &n);
8     for (i = 2; ((i * i) <= n) && (j == 0); i++)
9        if ((n % i) == 0)
10          j = 1;
11    printf("%d\n", j);
12 }
```

輸入

```
1 100
```

輸出

```
1 1
```

風格要點

一個迴圈只有一個入口及一個出口具有較高的可讀性。

continue

- 使用 `continue` 可跳過迴圈剩下的部分，繼續一個迴圈。
- 這通常是為了清楚表明程式的流程，表明在某種狀況下，我們必須重新回到迴圈的起點。
- 例如從讀入數字，但只挑選其中的正數相加。此時如讀到非正數，則不予處理並重新回到迴圈的起點。

範例程式 22: (continue.c) 挑選輸入的正數相加.

```
1  #include <stdio.h>
2  main()
3  {
4    int sum = 0;
5    int count, i, n;
6    scanf("%d", &n);
7    for (count = 0; count < n; count++) {
8      scanf("%d", &i);
9      if (i <= 0)
10       continue;
11     sum += i;
12   }
13   printf("%d\n", sum);
14 }
```

輸入

```
1 5
2 10
3 -30
4 24
5 -1
6 8
```

輸出

```
1 42
```

- 如果讀入的 i 不為正數，則我們立刻跳過迴圈剩下的部分，回到迴圈的頭。
- 讀者可以很清楚的了解 當讀入的 i 不為正數， **迴圈剩下的部分不用看**。
- 這對讀者是很重要的，因為他可以很清楚的知道，迴圈剩下的部分對不為正數的 i 是不重要的。

範例程式 23: (no-continue.c) 不使用 continue

```
1  #include <stdio.h>
2  main()
3  {
4    int sum = 0;
5    int count, i, n;
6    scanf("%d", &n);
7    for (count = 0; count < n; count++) {
8      scanf("%d", &i);
9      if (i > 0) {
10       sum += i;
11     }
12   }
13   printf("%d\n", sum);
14 }
```

- 當看到 `if (i > 0)` 時，並不知如果 `i <= 0` 時要如何處理。
- 如果 `if` 敘述很長，會看到最後面才恍然大悟原來 `i <= 0` 時 **不需要處理**。
- 沒有清楚表達 “只需考慮正數” 的想法。

輸入

```
1 5
2 10
3 -30
4 24
5 -1
6 8
```

輸出

```
1 42
```