1

上週內容

- 複合指定運算子
  - A `+=` 3;
  - B `-=` 4;
  - C `*=` 5;
  - D `/=` 2;
  - E `%=` 6;
- 運算子優先順序

if (…) { … } else { … }

if(…) { … } else if (…) {…} else if (…) {…}
…

2

3-4.cpp

```
#include <iostream>
using namespace std;
int main() {
  char ans;

  cout << "你今天高與嗎 (Y/N)? ";
  cin >> ans;

  if(ans == 'Y' || ans == 'y') {
    cout << "太好了 :-) \n";
  }
  cout  << "祝你有愉快的一天!\n";

  return 0;
}
```

```
你今天高與嗎 (Y/N)? n
祝你有愉快的一天!
```

```
你今天高與嗎 (Y/N)? p
祝你有愉快的一天!
```

```
你今天高與嗎 (Y/N)? Y
太好了 :-)
祝你有愉快的一天!
```

3

3-7.cpp

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

int main() {
  int score;

  cout << "請輸入成績: ";
  cin >> score;

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

  return 0;
}
```

4

3-9.cpp

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

int main() {
  int number;

  cout << "請輸入一個介於 1 - 10 之間的整數:";
  cin >> number;
```

3-9.cpp (續)

```
if(number >= 1 && number <= 10) {
   if(number%2 == 0) {
     cout << "你輸入的數字" << number << "是偶數";
   }
   else {
     cout << "你輸入的數字" << number << "是奇數";
   }
 }
 else {
     cout << "你輸入的數字" << number << "超出範圍";
 }

  return 0;
}
```


今日內容

- 浮點數的陷阱
- 流程控制 (II) – switch ... case ...
- 流程圖繪製
- 迴圈 (I) – for 迴圈

Lecture 4

流程控制 (II) switch ... case ...
流程圖繪製
迴圈 (I) – for 迴圈

浮點數的陷阱

- Recall: 電腦裡的浮點數是不精確的。
- 因此，在決策判斷 (if ... else ...) 的時候要小心
  - 在判斷兩浮點數是否相等時，通常我們會給予一容許誤差值

9

4-1.cpp

```
double a = 1.0, b = 1e-16, e = 1.0, f = 0.0;
float c = 16777216.0f;
int d = 0;

if (a == (a + b)) {
    cout << "\na == (a+b)";      a == (a+b)
}
if (a == (a - b)) {
    cout << "\na == (a-b)";      不會印出來 … Orz
}
if (c == (c + 1)) {
    cout << "\nc == (c+1)";      c == (c+1)
}
cout << "\n0/0==" << f / d;      0/0==-1.#IND    0/0==-nan
cout << "\n1/0==" << e / d;      1/0==1.#INF     1/0==inf
```

10

浮點數的陷阱

- 因此，在決策判斷 (if … else …)的時候要小心
  - 在判斷兩浮點數是否相等時，通常我們會給予一容許誤差值

```
if( (x – y) < abs(1e-8)) {
    … x 與 y 視為相等 …
}
```

11

switch … case

12

4-2.cpp

輸入年齡可以辨認適合看什麼級別的電影

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

int main() {
  int age;

  cout << "請輸入年齡: ";
  cin >> age;

  cout << "您可觀賞";
```

```
  switch(age/6) {
    case 0:
      cout << "普通級";
      break;
    case 1:
      cout << "保護級";
      break;
    case 2:
      cout << "輔導級";
      break;
    default:
      cout << "限制級";
  }

  return 0;
}
```

13

說明

- switch(...) { case ...} 為多重分支 (branching) 的另一種寫法，可用來取代if (...) {...} else if(...) {...} else (...) {...}。
- switch(A)，其中A
  - 需為整數(short, int, long) 或是字元型態 (char) 的**變數**。
  - 決定之後的那一個分支被執行到
- case B: ... break;
  - B 為上面A之值的可能值
  - 每一個 case 被稱為一個分支 (branch)
  - 程式在執行時會跳躍到檢查 A 變數當時的值來決定那一個分支被執行。若所有的 case 指定之 B 都與 A 之值不符合時，則程式會跳躍至 default (內定的) 分支。
  - break 很重要！switch() 只檢查一次參數，如果沒有 break 的話所有在符合條件之後的動作都會被執行到！

14

4-3.cpp

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

int main() {
  char grade;

  cout << "請輸入成績 (A, B, C, D, F): ";
  cin >> grade;

  switch(grade) {
    case 'A':
      cout << "90 - 100";
      break;
```

15

4-3.cpp

```
    case 'B':
      cout << "80 - 89";
      break;
    case 'C':
      cout << "70 - 79";
      break;
    case 'D':
      cout << "60 - 69";
      break;
    default:
      cout << "0 - 59";
  }
  return 0;
}
```

16

玩玩看

- 如果把4-2.cpp 中的 break; 拿掉，程式是否可以執行？結果是否正確？
- 同樣的，如果把 4-3.cpp 中的 break; 拿掉，你是不是可以預測它的結果？
- 4-3.cpp 只能輸入大寫字母；如何修改使其大小寫字母都可以接受？

17

Fall through

- 在執行 switch (...) {case ...} 時，程式的分支判斷只執行一次，而一支分支的終點以 break; 決定之。

```
case 'B':
     cout << "80 - 89";
     break;
case 'C':
     cout << "70 - 79";
     break;
```

```
case 'B':
     cout << "80 - 89";
case 'C':
     cout << "70 - 79";
     break;
```

- 若一分支沒有 break 的話，接下來的分支亦會被執行，直到碰到 break 或是整個 switch 的區塊結束。
- switch case 中的 case 實質上只是一個標籤，而分支的動作是由 switch 執行的！！！

18

4-3a.cpp (改自 4-3.cpp)

```
switch(grade) {
  case 'A': case 'a':
    cout << "90 - 100";
    break;
  case 'B': case 'b':
    cout << "80 - 89";
    break;
  case 'C': case 'c':
    cout << "70 - 79";
    break;
```

```
  case 'D': case 'd':
    cout << "60 - 69";
    break;
  default:
    cout << "0 - 59";
}
```

19

if ... 和 switch ... 效率比較

- 在多重的 if ... else if ... 中，若判斷條件是簡單的 == 時，switch ... case ... 和 if ... else if ... 常常可以互相取代的。
- switch case 在編譯時，實際上是產生跳躍表 (jump table)，根據 switch 的值直接跳躍到對應的程式碼。
- if ... else if ... else if ... 則是在執行時期一個條件一個條件的檢查，所以執行上的效能會相對較差。

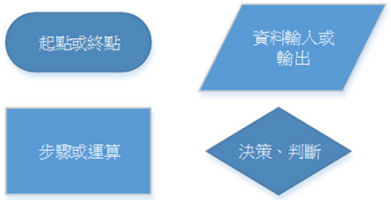
20

流程圖的繪製

21

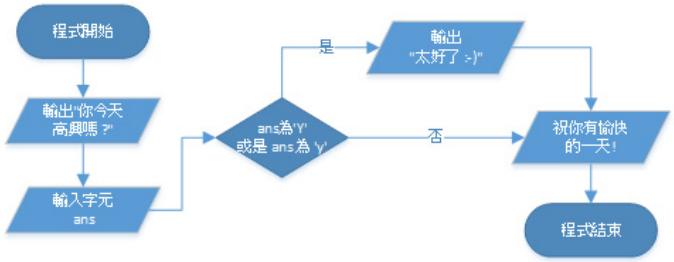
流程圖 (flowchart)

- 用來設計與展示**步驟與流程**的圖形，不僅限應用於程式設計
- 四個基本形狀、之間以箭頭連結以代表先後關係
- 在初學程式寫作時，每個矩形內寫入一個敍述即可;
- 所有的形狀可有一至多個進入點；但除了菱形外都只有一個離開點。
- 菱形具多個離開的路徑 (分支點)，每個路徑上需註明選擇該路徑的條件。



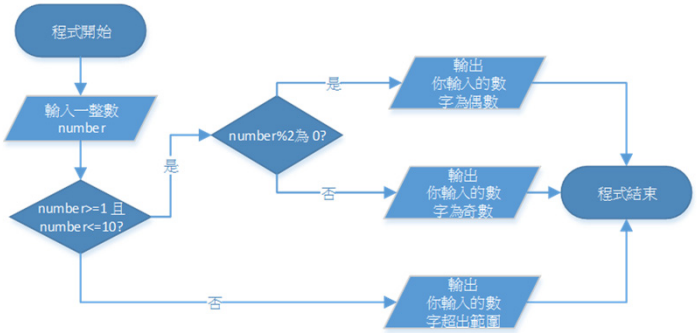
22

以 3-4.cpp 為例



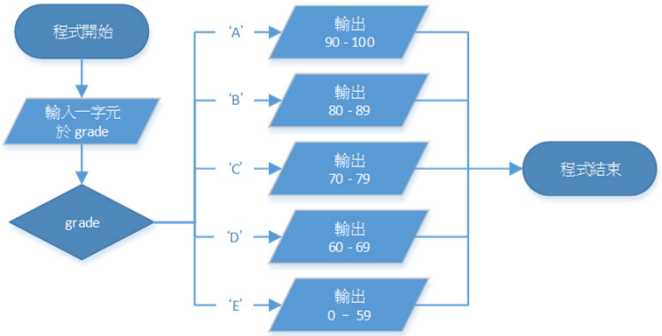
23

以 3-9.cpp 為例



24

以 3-7.cpp為例



25

繪製工具

- Microsoft Visio (非免費、需安裝)
- Dia: https://wiki.gnome.org/Apps/Dia (免費、有中文、需安裝)

  Dia 的教學
  http://elesson.tc.edu.tw/md221/course/view.php?id=47

- 線上直接使用的:
- https://cacoo.com/
- https://www.lucidchart.com/

26

繪製流程圖的好處

- 與程式語言無關，可以和不會寫程式的人討論
- 流程圖可用來檢討所設計的步驟流程是否合理、是否能正確解決問題
- 待流程確認後，才著手進行程式的撰寫，可以減少程式邏輯發生錯誤的機會。

27

迴圈 (loop)

- 迴圈的用途
  - 用來重複執行某一段程式 → 執行**重複性**的事務
  - 之後會介紹到配合陣列 (array) 以對大量資料進行處理 (其實，就是重複性的計算)

- C/C++ 的迴圈 (loop) 指令
  - for(...;...;...) {...} – 多應用於具固定次數的重複性
  - while(...) { ... } – 應用於重複次數不固定的場合
  - do {...} while(...); – 應用於重複次數不固定的場合

28

for 迴圈

29

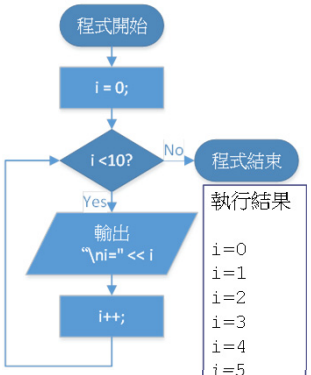
4-4.cpp

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

int main() {
  int i;

  for(i=0; i<10; i++) {
      cout << "\n i=" << i;
  }

  return 0;
}
```



執行結果

```
i=0
i=1
i=2
i=3
i=4
i=5
i=6
i=7
i=8
i=9
```

30

for 的語法

- for (初始運算式；條件運算式；控制運算式) {
  迴圈內容或欲重複執行的程式片段;
  …
  }
- 初始運算式：迴圈內容開始執行前執行的敍述
- 條件運算式：條件運算式在 1) 迴圈執行前 和 2) 迴圈內容完成後執行，若運算結果為真，才會執行或是再執行迴圈的內容
- 控制運算式：迴圈內容完成一次後執行的敍述
- 以上三個運算式都可以省略 → 無窮迴圈，迴圈內容會不斷地重複執行

31

for 的語法

- for(i=0; i<10; i++) { cout << "\i=" << i; }
  - i = 0: 「先」設定 i 變數的值為 0
  - i < 10: 檢查迴圈的內容是否**要執行**或是否要**繼續執行**？
  - i++: 當迴圈內容作完一次後要執行的動作
  - i 為迴圈的控制變數，通常稱之為計數器 (counter)。

32

for 迴圈

```
    for (i=0; i<3; i++) {
4       cout << "a";
    }

5   cout << "b";
```

1 2 3

1 迴圈被執行到時初始化的動作
2 迴圈要執行或要被繼續執行的條件
3 迴圈完成一次後要做的動作
4 迴圈內欲重複執行的敍述
5 迴圈結束後要執行的敍述

1	令 i = 0
2	0 < 3 → *true*
4	cout << "a";
3	i++;
2	1 < 3 → *true*
4	cout << "a";
3	i++;
2	2 < 3 → *true*
4	cout << "a";
3	i++;
2	3 < 3 → ***false***
5	cout << "b";

33

4-5.cpp

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

int main() {
  int i;

  for(i=0; i<10; i=i+2) {
      cout << "\n i=" << i;
  }

  return 0;
}
```

i+=2

執行結果

```
i=0
i=2
i=4
i=6
i=8
```

34

4-6.cpp

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

int main() {
  int i;

  for(i=1; i<10; i=i+2) {
      cout << "\n i=" << i;
  }

  return 0;
}
```

執行結果

```
i=1
i=3
i=5
i=7
i=9
```

35

練習

- 請填入以下程式空白處，使程式結果輸出如下數列。

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

int main() {
  int i;

  for(i=_____; _____; _____ ) {
    cout << i << ", " ;
  }
  cout << endl;

  return 0;
}
```

10, 11, 12, 13, 14, 15, 16, 17, 18, 19,

10, 12, 14, 16, 18, 20, 22, 24, 26, 28,

5, 10, 15, 20, 25, 30, 35, 40, 45,

45, 40, 35, 30, 25, 20, 15, 10, 5,

-10, -7, -4, -1, 2, 5, 8,

1, 2, 4, 8, 16, 32, 64,

36

4-7.cpp

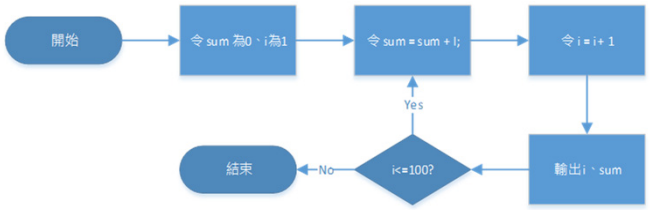
- 如何使用電腦幫我們計算從 1 加到 100 的總和？
  - 這裡，我們要先教電腦笨的方法 (聰明的方法是什麼？)
  - 記得，如果你不會算，程式大概就寫不出來了 ...
  - 所以，你在寫程式的時候，一定要想想自己怎麼作一件事情，而且要把步驟拆得很細，因為電腦很笨 ...

37

4-7.cpp 事前規劃

- 有沒有輸入？
- 有什麼樣的輸出？
- 要怎麼從輸入或已知道的資料得到輸出結果？

```
i=1, 總和=1
i=2, 總和=3
   .
   .
   .
i=95, 總和=4560
i=96, 總和=4656
i=97, 總和=4753
i=98, 總和=4851
i=99, 總和=4950
i=100, 總和=5050
```



38

4-7.cpp

```
i=1, 總和=1
i=2, 總和=3
   .
   .
   .
i=95, 總和=4560
i=96, 總和=4656
i=97, 總和=4753
i=98, 總和=4851
i=99, 總和=4950
i=100, 總和=5050
```

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

int main() {
  int i, sum=0;

  for(i=1; i<=100; i++) {
      sum = sum + i;
      cout << "\ni=" << i << ", 總和=" << sum;
  }

  return 0;
}
```

39

注意事項

- 寫迴圈時記得將迴圈的內容縮排 (內縮, indent), 用來便於識別那一部份是迴圈的內容!
  - 寫巢狀式的 if 或 switch-case 時 ，也應養成縮排的好習慣!!
- 如果沒有**很好的理由**, 不要在 for-loop 迴圈的區塊內改變迴圈計數器的值, 否則在除錯時會很困難...

40

for 的語法

- 迴圈的內容執行完一次，回到控制運算式，稱作一個**迭代** (iteration)
- for 迴圈又被稱作計數器控制迴圈 (counter-controlled loop)，通常都會利用一個或一個以上的變數作為「計數器」，用來計算並控制迴圈的迭代次數。有時，會利用此計數器變化每次迭代所作的計算。
  `for(i=0;i<123;i++) { … }`
- 三個運算式中初始運算式 (e.g. i=0) 與控制運算式 (e.g. i++) 可以有一個以上的運算，並以逗點隔開。其中條件運算部份不宜使用逗點，而應使用邏輯運算子複合你想要表達的迴圈執行條件。

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

41

4-8.cpp

- 若從今天起，第一天存 1 元，每一天就存入前一天存入錢的兩倍 (e.g. 1, 2, 4, 8, 16, 32, …)，請問到第 30 天結束後，你總共存入了多少錢？

42

4-8.cpp

```
第 1 天: 存入 1 元, 共有 1 元
第 2 天: 存入 2 元, 共有 3 元
第 3 天: 存入 4 元, 共有 7 元
第 4 天: 存入 8 元, 共有 15 元
第 5 天: 存入 16 元, 共有 31 元
第 6 天: 存入 32 元, 共有 63 元
第 7 天: 存入 64 元, 共有 127 元
第 8 天: 存入 128 元, 共有 255 元
第 9 天: 存入 256 元, 共有 511 元.
.
.
第 28 天: 存入 134217728 元, 共有 268435455 元
第 29 天: 存入 268435456 元, 共有 536870911 元
第 30 天: 存入 536870912 元, 共有 1073741823 元
```

43

4-8.cpp

```
第 1 天: 存入 1 元, 共有 1 元
第 2 天: 存入 2 元, 共有 3 元
第 3 天: 存入 4 元, 共有 7 元
第 4 天: 存入 8 元, 共有 15 元
第 5 天: 存入 16 元, 共有 31 元
第 6 天: 存入 32 元, 共有 63 元
第 7 天: 存入 64 元, 共有 127 元
第 8 天: 存入 128 元, 共有 255 元
第 9 天: 存入 256 元, 共有 511 元.
.
.
第 28 天: 存入 134217728 元, 共有 268435455 元
第 29 天: 存入 268435456 元, 共有 536870911 元
第 30 天: 存入 536870912 元, 共有 1073741823 元
```

```cpp
#include <iostream>
using namespace std;

int main() {
    int saving = 1;
    int total = 0;

    for(int i=1;i<=30;i++) {
        total = total + saving;
        cout << "第 " << i << " 天: ";
        cout << "存入 " << saving << " 元, ";
        cout << "共有 " << total << " 元" << endl;
        saving = saving * 2;
    }
    return 0;
}
```

如果現在想要知道 60 天後的結果呢？

44

4-9.cpp

請列出 1 – 50 之間，可以被 3 或可以被 5 整除的所有數字，並在最後輸出有幾個符合條件的數字。

4-9.cpp

- 變數在使用前宣告即可。
- 此處的變數 *i* 僅在 for 迴圈內有被定義，以此範例中，*i* 變數只有在緊接的 {...} 內看得到，到了最後的 cout 處其實 *i* 變數沒有被定義。

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

int main() {
    int count=0;
    for(int i=1;i<=50;i++) {
        if(i%3 == 0 || i%5 == 0) {
            cout << i << ", ";
            count++;
        }
    }
    cout << "\n" << count << endl;
    return 0;
}
```


隨堂練習

請撰寫一程式，讓使用者依序輸入一底數 (base) a 、以及指數 (exponent) n，並輸出乘冪 (power) a^n 為多少。

```
請輸入底數 a: 2.25
請輸入指數 n: 3
乘冪 a^n 為 11.3906
```

```
請輸入底數 a: 10
請輸入指數 n: 3
乘冪 a^n 為 1000
```


作業三

目前的西元曆法，為西元1582年開始使用的格列哥里曆 (Gregorian calendar)，在此曆法下，每400年設置了97個閏年，以降低曆法與實際地球對太陽公轉的偏差。其閏年設置的規則以西元年份作為判斷：

原則上可被 4 整除的那年為閏年 (1604, 1608, 1612, ...)

但若可以被100整除的那年不為閏年 (1700, 1800, 1900, 2100)

但若可以被400整除的那年又是閏年 (1600, 2000, 2400, ...)

請撰寫一程式，讓使用者輸入開始年與結束年，並列印出期間 (包含開始年與結束年) 的所有閏年，列印時每列最多印十個年份，並在最後輸出該期間總共有幾個閏年。

https://en.wikipedia.org/wiki/Gregorian_calendar
http://pansci.asia/archives/94403