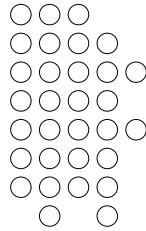


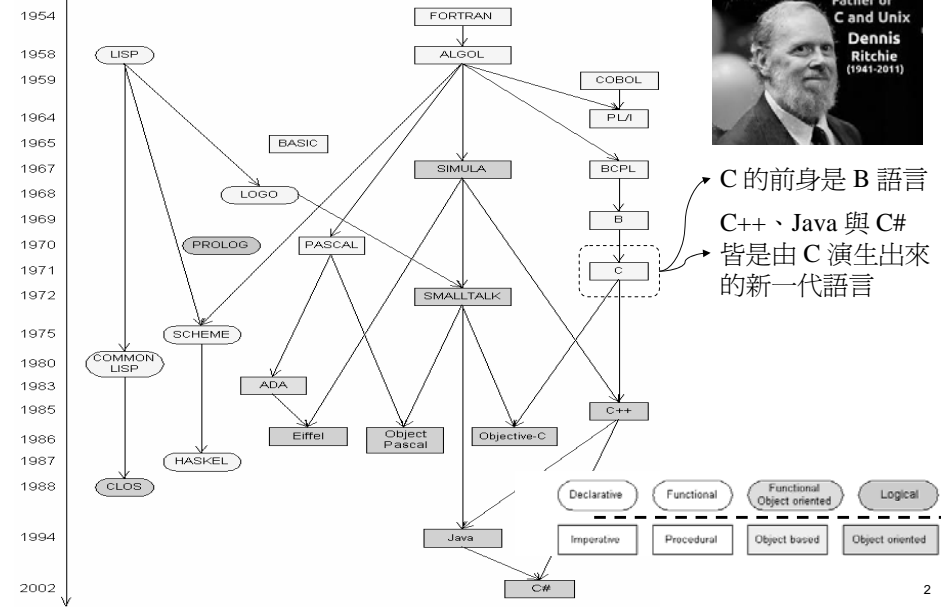
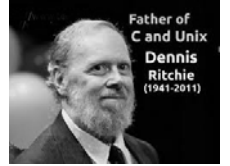
第一章 認識 C 語言

C 語言的歷史
程式的規劃與實作
第一個 C 程式
程式碼的編譯與執行



高階程式語言的發展歷史

1.1 認識C語言



1.1 認識C語言

Declarative Language



- also called nonprocedural or very high level language
- describes **what** it wants to accomplish rather than **how** to achieve its goal
- e.g.
 - SQL: specify the property of the result data
Select Region, Profit from Sales where Profit > 700
 - Prolog: answer the whole maternal family tree with logic deduction
ancestor(M, C) :- mother(M, C)
ancestor(X, Z) :- mother(X, Y), ancestor(Y, Z)

1.1 認識C語言

Imperative Language



- uses a sequence of statements to specify exactly the procedure for obtaining a certain result
- e.g.


```
int total = 0;
int number1 = 5;
int number2 = 10;
int number3 = 15;
total = number1 + number2 + number3;
```

Each statement changes the state of the program, from assigning values to each variable to the final addition of those values. Using a sequence of five statements the computer is explicitly told how to add the numbers 5, 10 and 15 together – based on the computer's architecture.

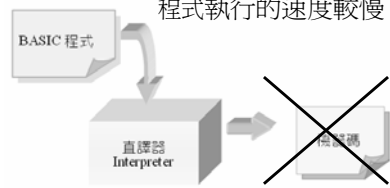
C 語言的特色 (1/4)

● 高效率的編譯式語言

- C 為編譯式語言，執行速度遠比直譯式快

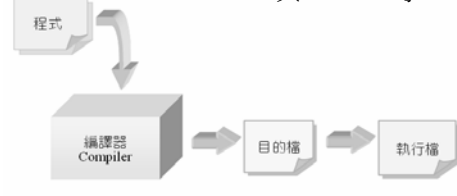
直譯器 (interpreter)

- 一次解釋一個敘述，然後在直譯器中以“等效”的方式直接執行它，直到結束敘述
- 測試、修改程式很快很方便，但程式執行的速度較慢



編譯器 (compiler)

- 將整個程式碼編譯成機器碼，然後由 CPU 直接執行
- 常見的編譯式程式語言有 C、C++、Fortran 與 Pascal 等



5

直譯器 vs. 編譯器

● 假設老師給你一個資料統計的工作:

- 由整理輸入資料一直到最後作出結果有多個步驟
- 處理過程中間的數值並不要求你繳交
- 大多數同學應該都會用自己最喜歡的方式記錄中間的資料，甚至有的時候會合併某些計算步驟，如此可以比較快速地得到結果...
- 壞處: 萬一你最後的結果計算錯了，老師沒有辦法根據你的中間計算資料來判斷錯誤的原因，也沒有辦法在一開始發生問題時趕快停下來

- 編譯器: 一次把整個程式翻譯成機器可以有效率執行的低階命令，中間處理的結果都以能有效率繼續處理的方式儲存而不輸出
- 直譯器: 一系列程式逐步解釋，逐步模擬執行，並且讓執行的人隨時可以中斷或是繼續執行，隨時可以看到中間的處理結果

6

C 語言的特色 (2/4)

● C 語言兼具

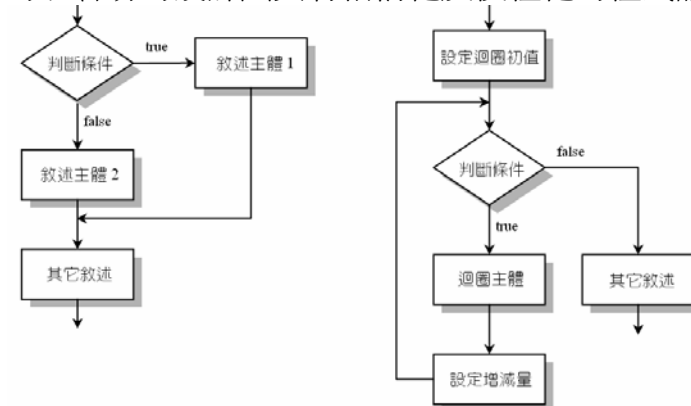
- 高階語言的優點
- 低階語言的特色
 - 低階語言如組合語言與機器碼，直接控制電腦執行
 - 高階語言貼近人類語言，如 BASIC、FORTRAN，適合人類閱讀

7

C 語言的特色 (3/4)

● 完整的流程控制與結構化語法

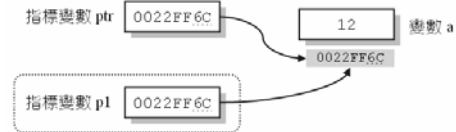
- 可以容易的設計出具有結構化及模組化的程式語言



8

C 語言的特色 (4/4)

- 可攜性佳
 - 想跨越平台 (Windows, Mac OS, Unix, ...) 來執行C語言，通常只要修改極少部分的程式碼，再重新編譯即可執行
- 貼近底層計算機模型的語言
 - C語言可以直接依記憶體位址來存取變數，以提高程式執行的效率



9

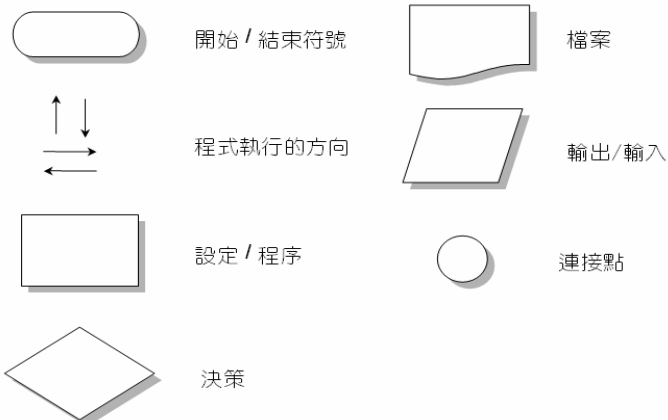
C 語言與其它程式語言的關係

- 程序式 (procedural) 語言 (Basic, Fortran, Pascal, C) 都具有共通的流程控制語法
- C++, Objective C, 與 Java 都包含 C 的流程控制語法，再加上額外支援物件導向 (Object Oriented) 的語法，使得它們可以用物件導向的方式設計視窗圖形介面、網路應用、並且可以用來設計大型應用軟體

10

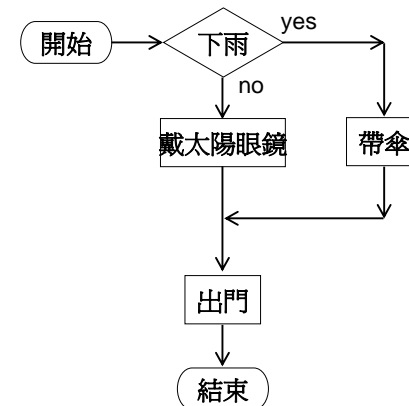
流程圖符號

- 下面為繪製流程圖時常用的符號：



11

流程圖繪製的範例



如果下雨，則帶傘，否則戴太陽眼鏡。不管是否下雨，最後都要出門

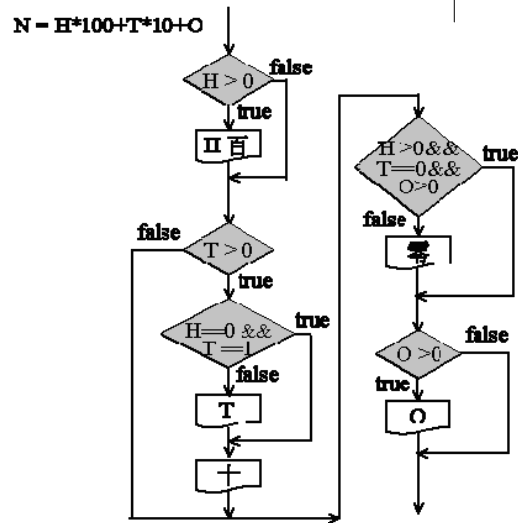
Flowchart Guide: <http://creately.com/blog/diagrams/flowchart-guide-flowchart-tutorial/>
Common Mistakes:

1. <http://creately.com/blog/diagrams/part-1-15-mistakes-you-would-unintentionally-make-with-flowcharts/>
2. <http://creately.com/blog/diagrams/part-2-15-mistakes-you-would-unintentionally-make-with-flowcharts/>

12

流程圖的使用時機

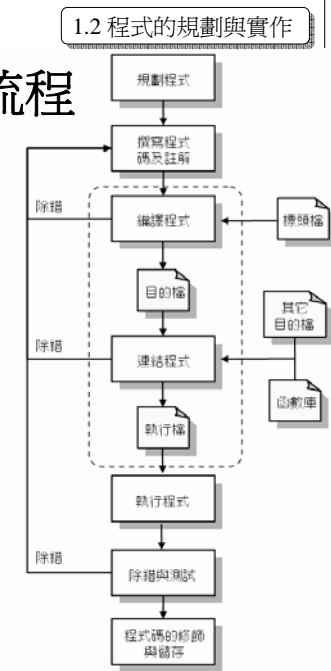
- 寫程式一定要畫流程圖嗎?
- 是先畫圖再寫程式還是先寫程式再畫圖?



13

程式規劃與實作的流程

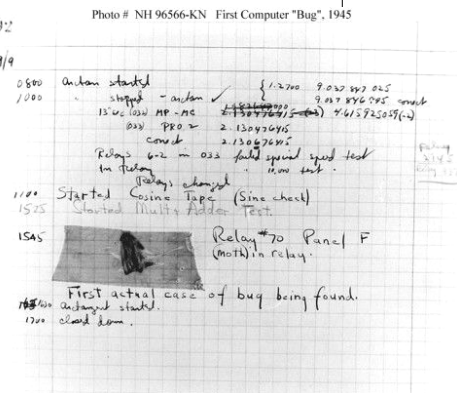
- 規劃程式
- 撰寫程式碼及註解
- 編譯程式碼
- 執行程式
- 除錯與測試
- 程式碼的修飾與儲存



14

程式的錯誤

- 語意錯誤 (semantic error)
 - 程式的執行結果不如預期
 - 邏輯錯誤
- 語法錯誤 (syntax error)
 - 程式中有不合語法的敘述
- 找出錯誤的過程稱為除錯 (debug)



記載電腦 bug 的維修記錄本

本圖片轉載自 <http://www.computersciencelab.com>

15

1.3 撰寫第一個C程式

第一個 C 程式

- 以 Dev C++ 環境撰寫第一個 C 程式：

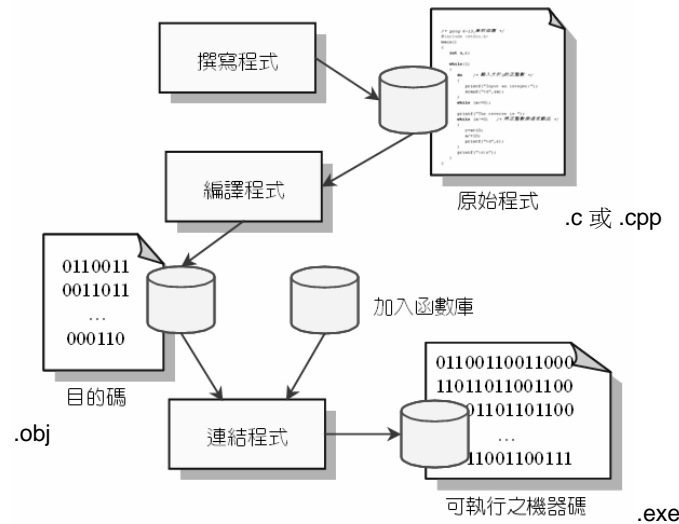
```

01  /* prog1_1, 第一個 C 程式碼 */
02  #include <stdio.h>
03  #include <stdlib.h>
04  int main(void)
05  {
06      printf("Hello C!\n");      /* 印出 Hello C! 字串 */
07      printf("Hello World!\n"); /* 印出 Hello World! 字串 */
08
09      system("pause");
10      return 0;
11  }
    
```



16

原始程式編譯及連結的過程



17

高階 C vs. 低階組合語言(Assembly)

```

01 # include <stdio.h>
02 # include <stdlib.h>
03 int main(void)
04 {
05     int i;
06     for (i=0; i<10; i++)
07         printf("i=%d\n", i);
08     system("pause");
09     return(0);
10 }

```

```

; Line 6
        mov     DWORD PTR _i$[ebp], 0
        jmp     SHORT $L901
$L902:
        mov     eax, DWORD PTR _i$[ebp]
        add     eax, 1
        mov     DWORD PTR _i$[ebp], eax
$L901:
        cmp     DWORD PTR _i$[ebp], 10
        jge     SHORT $L903
; Line 7
        mov     ecx, DWORD PTR _i$[ebp]
        push   ecx
        push   OFFSET FLAT:$SG904
        call   _printf
        add     esp, 8
        jmp     SHORT $L902
$L903:

```

18

與組合語言緊密結合

• Inline Assembly

- `asm("movl %ecx %eax"); /* moves the contents of ecx to eax */`
- `__asm__("movb %bh (%eax)"); /* moves the byte from bh to the memory pointed by eax */`
- `__asm__ ("movl %eax, %ebx\n\t"
"movl $56, %esi\n\t"
"movl %ecx, $label(%edx,%ebx,$4)\n\t"
"movb %ah, (%ebx)");`
- `int main(void) {
int foo = 10, bar = 15;
__asm__ __volatile__ ("addl %%ebx, %%eax"
:"=a"(foo) // result in %eax, =: output register
:"a"(foo), "b"(bar); // store foo in %eax, bar in %ebx
printf("foo+bar=%d\n", foo);
return 0;
}`

結構化 vs. 非結構化

- 12,3,37,8,24,15,5,33 → 3,5,8,12,15,24,33,37

由小至大排列

```

void selectionSort(int data[], int ndata) {
    for (int i=0; i<ndata; i++)
        findMinimumOfAnArray(&data[i], ndata-i);
}
void findMinimumOfAnArray(int data[], int ndata) {
    int min = 0;
    for (int i=1; i<ndata; i++)
        if (data[i]<data[min]) min = i;
    swap(&data[0], &data[min]);
}
void swap(int *x, int *y) {
    int tmp = *x; *x = *y; *y = tmp;
}

int d[] = {12, 3, 37, 8, 24, 15, 5, 33};
int n = 8; *d1, *d2, *p, *e;
d1 = d; d2 = d+n;
l1: if (d1>=d2) goto end;
p = d1;
e = d1 + 1;
l2: if (e>=d2) goto next;
if (*e<*p) p = e;
e++;
goto l2;
next: n = *p; *p = *d1; *d1 = n;
d1++;
goto l1;
end:

```

20