Fortran Week 4

- 本週進度補充
  - 自訂函數、副程式
  - Whole array operation
  - 運算效率
  - Debug

- HW12解答、HW13說明

補充(1)：主、副程式的變數傳遞

- FORTRAN在主、副程式之間傳遞變數時，是傳遞記憶體的位置，而不是變數名稱
- 主程式呼叫時給予的變數清單（arg_list），只要變數種類、個數相符，就會按照arg_list的順序傳入副程式，就算變數名稱不一致也沒關係。舉例：

（主程式）

```
PROGRAM  main
IMPLICIT NONE
INTEGER :: a=1, b=0
CALL sub1(a,b)
WRITE(*,*) 'in main',a,b
END PROGRAM main
```

執行結果 →

```
in sub 1 0
in main 1 99
```

（副程式）

```
SUBROUTINE sub1(x,y)
INPLICIT NONE
INTEGER,intent(in) :: x
INTEGER,intent(inout):: y
WRITE(*,*) 'in sub', x,y
y=99
RETURN
END SUBROUTINE sub1
```

補充說明(1): Subroutine/Function變數宣告 intent(in)? (out)? (inout)?

- 簡單的規則，根據變數在副程式內
- 只出現在等號右邊，或在宣告區決定陣列大小 → intent(in)
  - 這個變數的值從主程式傳進來，目的是為了用來替其他變數賦值
- 第一次出現的時候在等號左邊 → intent(out)
  - 這個變數的值在副程式計算出來，回傳給主程式
- 第一次出現的時候在等號右邊，之後也有出現在左邊 → intent(inout)
  - 這個變數的值從主程式傳進副程式，有用來為其他變數賦值，而且它的值也有在副程式中被變動，變動後的結果要回傳給主程式

補充(2)：主程式與自訂函數(或副程式)要放在同一個檔案中?還是分開?

- 都可以，看需求而定

Q: 哪種情況下，適合把主程式與自訂函數(或副程式)寫在同個.f95檔?

A: 自訂函數(副程式)只會在這個主程式用到，不會被其他主程式使用

(所謂“一次性”程式)

Q: 那為什麼還要花力氣寫自訂函數(副程式)?

A: 可以讓主程式簡潔清楚

Q: 寫在同一個.f95的主程式+自訂函數怎麼編譯?

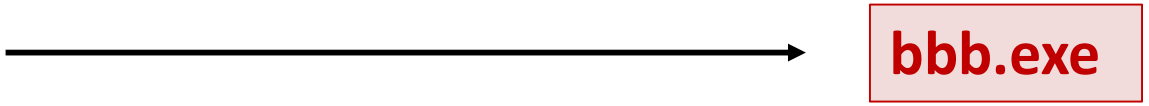
A: 跟編譯主程式一樣

補充(2)：主程式與自訂函數(或副程式)在同一個檔案中的編譯方法

(aaa.f95)

```
Program XXX

... (主程式在這)

END PROGRAM XXX


Function  YYY
... (自訂函數在這)

RETURN
END FUNCTION YYY
```

f95 aaa.f95 -o bbb.exe



bbb.exe

補充(2)：主程式與自訂函數(或副程式)要放在同一個檔案中?還是分開?

Q: 哪種情況下，適合把主程式與自訂函數(或副程式)分開寫在不同的.f95檔?

A: 自訂函數(副程式)有機會被其他主程式用到，是個應用性很高的工具

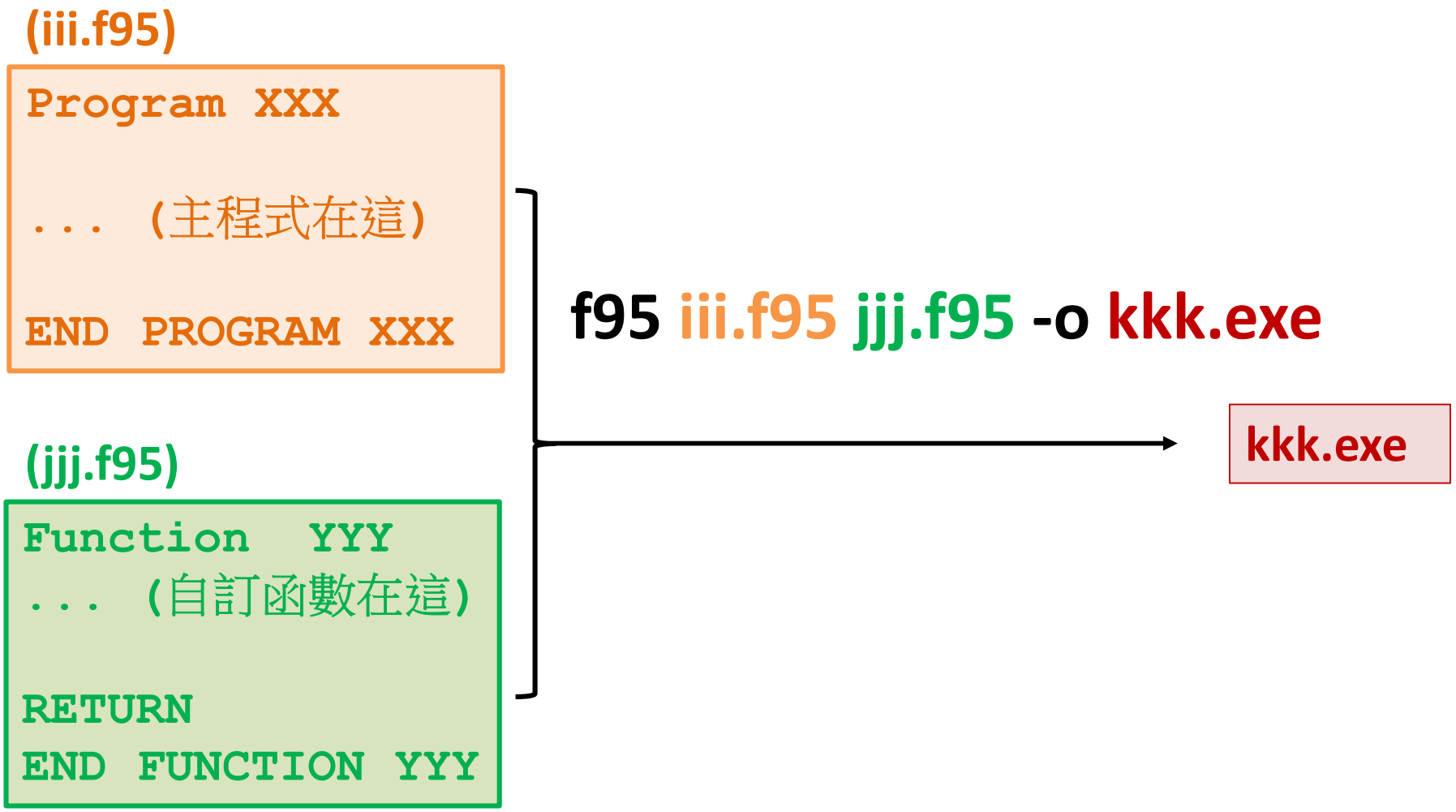
Q: 分開寫自訂函數(副程式)有甚麼好處?

A: 以後要用到不需要重寫或重複貼到新的.f95檔案中，直接透過編譯就可以”隨插即用“，也可以讓主程式簡潔清楚

Q: 分開在不同.f95檔的主程式+自訂函數怎麼編譯?

A: 把自訂函數(副程式)的.f95檔列在主程式.f95檔的後面

補充(2)：主程式與自訂函數(或副程式)在不同檔案的編譯方法



補充(2)：主程式與自訂函數(或副程式)在不同檔案的編譯方法

(mmm.f95)

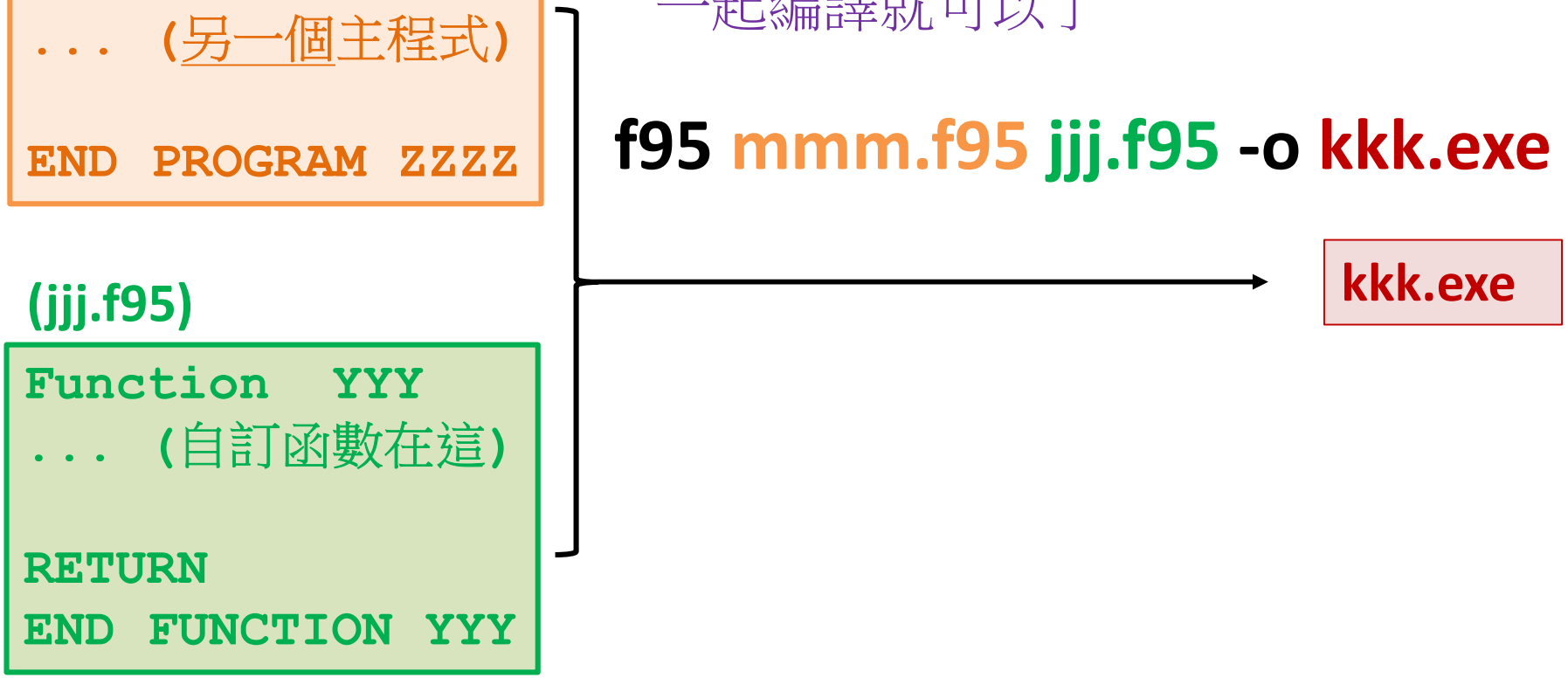
```
Program ZZZZ

... (另一個主程式)

END PROGRAM ZZZZ
```

(jjj.f95)

```
Function  YYY
... (自訂函數在這)

RETURN
END FUNCTION YYY
```

如果下次有另一個主程式要用到同一個自訂函數，只要寫好主程式再一起編譯就可以了

f95 mmm.f95 jjj.f95 -o kkk.exe



補充(2)：主程式與自訂函數(或副程式)在不同檔案的編譯方法

- 編譯指令中，執行檔名稱 **-o kkk.exe** 與程式檔案清單 **iii.f95 jjj.f95** 順序可以調換
  (下面兩種寫法都可以)

> **f95 iii.f95 jjj.f95 -o kkk.exe**

> **f95 -o kkk.exe iii.f95 jjj.f95**

- 但 **-o** 後面要緊接著執行檔名 **kkk.exe**

- 把編譯指令寫成shell script(純文字檔)

(runcode.sh)

```
#!/bin/bash
f95 pr4b_main.f95 pr4b_sub.f95 -o pr4b.exe
./pr4b.exe
```

第一行指定要使用的shell介面
(這邊用bash shell當例子)

第二行開始，把想要在UNIX/LINUX命令列執行的指令依序列出來，一行一個指令，可以有很多行

- 在UNIX/LINUX命令列，更改shell script的權限為"可執行"

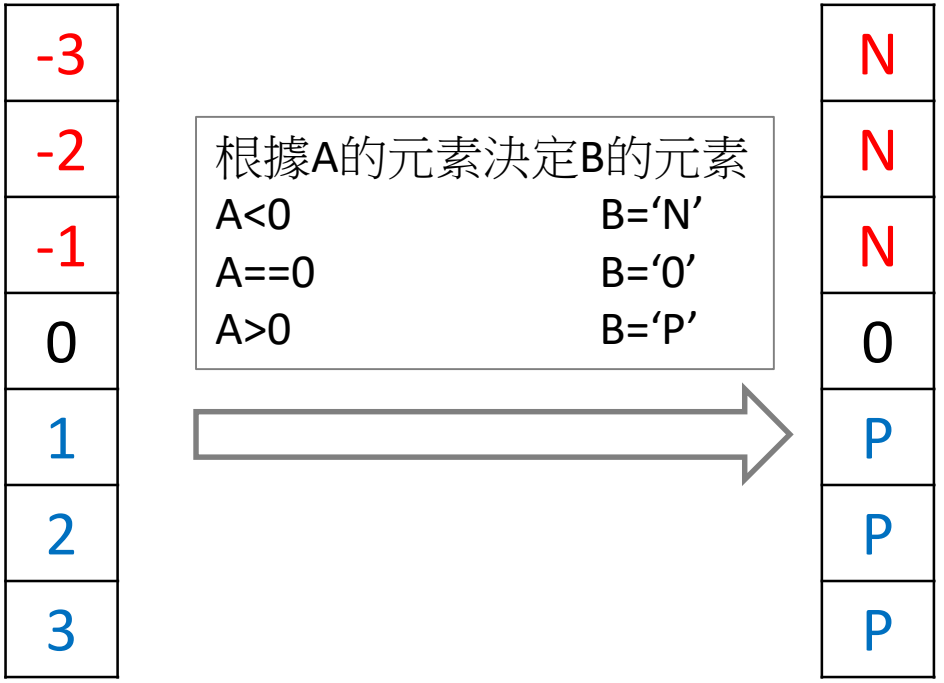
```
> chmod +x runcode.sh
```

- 在UNIX/LINUX命令列，執行shell script就可以完成編譯與執行

```
> ./runcode.sh
```

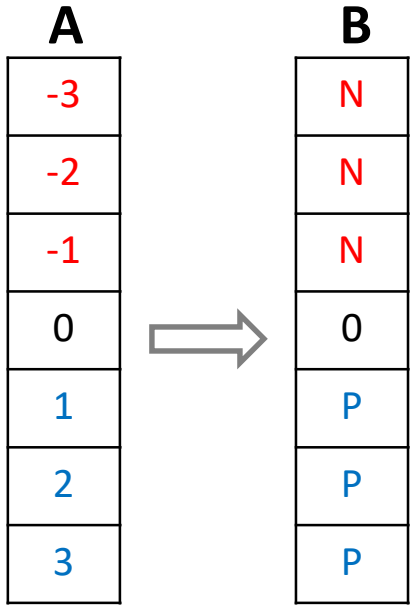
補充(3)：用where指令"篩選"陣列

- 假設有A, B兩個相同大小的陣列
- **WHERE**可以根據A陣列的元素數值設定條件，為B陣列同位置的元素賦值(類似mask功能)
- E.g. **integer, dimension(7)::A** **character(1), dimension(7)::B**



- 用迴圈搭配判斷式依序處理個別元素 (慢)

```
Do i=1,7
   IF (A(i)<0) THEN
     B(i)='N'
   ELSEIF (A(i)==0) THEN
     B(i)='0'
   ELSE
     B(i)='P'
   END IF
END DO
```



- 用where處理所有符合條件的元素(快)

```
WHERE (A<0) B='N'
WHERE (A==0) B='0'
WHERE (A>0) B='P'
```

補充(4) 使用內建函數的運算速度較快(以dot_product為例)

- Fortran內建函數在編譯程執行檔後，可運用CPU自身提供的運算功能，通常運算速度會比自己撰寫程式碼要快
- 如果運算過程中會”使用迴圈對陣列元素逐一做運算”(例如內積)，會更明顯感受到運算速度的差異
- 要盡可能多使用whole array operation與陣列內建函數，避免使用多層迴圈搭配if，才能有更快的運算效率

- 範例/home/teachers/weitingc/lecture_ex/inner_product.f95 其中用到兩個fortran內建函數
  - CALL RANDOM_NUMBER(u) (用亂數為實數陣列u賦值,）
    CALL CPU_TIME(start) (以目前的CPU時間為實數變數start賦值，可用來對程式執行計時)

補充(5)：運算速度Fortran vs Python

- 之前也測試過Python 「一維陣列內積」的運算時間，比較內建函數與直接迴圈計算的速度

  ~/weitingc/lecture_ex/innerproduct.py

  Python Numpy內建 c1=np.dot(a,b) (0.1 s)

  Python 簡單迴圈 for i in range(n): c2=c2+a[i]*b[i] (82 s!!)

- FORTRAN 內建 (0.15 s) vs. FORTRAN 迴圈 (0.56 s)

→

FORTRAN intrinsic function ~ Python Numpy intrinsic >

FORTRAN Loop >> Python Loop

補充(5)：運算速度Fortran vs Python

- 在編譯FORTRAN的時候，進行不同等級的最佳化optimization(-O1~-O3等)，可進一步加快FORTRAN loop的運算:

- f95 **-O1** inner_product.f95 -o inner_product_O1.exe
  - FORTRAN using intrinsic= 0.14
  - FORTRAN using loop= 0.20 → 變快了!

補充(5)：運算速度Fortran vs Python

- 結論：在處理大量數值資料的時候，優先使用FORTRAN(盡量用whole array operation與內建函數，或編譯時最佳化)，
- Python Numpy的whole array與內建函數也很快!
- 避免使用Python迴圈處理陣列個別元素—很慢!!

- 還想更快? → 平行化!!
  - 把運算分配給不同的CPU核心，同時進行(多管齊下)
  - 目前主流的平行化工具: MPI
    - 都有支援Fortran與python (mpi4py)的函式庫
    - 未來有機會在雲動力、雲與環境等進階選修課實際用到

補充(5)：運算速度Fortran vs Python

- 為什麼大氣系要學Fortran？
  - 運算速度快、高度支援平行運算
  - 現有的天氣預報模式、氣候模式絕大多數為fortran程式
    - 歷史因素(許多模式在70年代開始發展雛形，當時主流使用FORTRAN 77)
    - 需要在大量的網格上進行流體的運動方程計算
      →非常需要高效能的平行運算來加快速度

補充(6)：Debug demo

/home/teachers/weitingc/work/
demo_bug.f95, data5.txt

- 皮卡丘進行了雲滴成長的實驗，量測雲滴體積每0.025秒的增長率，**data5.txt** 是他的測量結果，單位是 ‰/0.025s，共有45筆量測數值。
- **demo_bug.f95**是皮卡丘寫的程式，想要計算出這45筆增長率的「幾何平均」(geometric mean），並且輸出實驗時間在第0.5秒時的量測結果。但是程式卻無法得到想要的結果:

  data at 0.5 sec= 100.263298

  geometric mean= 152.061081

- 若有n個數值$\{x_1, x_2, x_3 \ldots x_n\}$，其「幾何平均」定義為

$$\left(\prod_{i=1}^{n} x_i\right)^{\frac{1}{n}}$$

即 $\sqrt[n]{x_1 \cdot x_2 \cdot \cdots \cdot x_n}$

- 編譯並執行demo_bug.f95 ，症狀:
  - (1)無法輸出時間0.5秒的量測結果（判斷式Tt==0.5從未為真!?）。
  - (2)幾何平均結果（變數gaP）為Infinity

- 在Tt=Tt+dt這行之後，把k, dt與Tt的數值顯示到螢幕觀察
  - →症狀1是數值誤差造成
  - 解決方法：判斷式由Tt==0.5改成abs(Tt-0.5)<1.E-6

- 在Tt=Tt+dt這行之後，把k, P與F的數值顯示到螢幕上觀察
  - →症狀2是溢位造成
  - 解決方法：先整體縮小P的值（如/1000.)，計算出幾何平均後再還原 (*1000)