Maitrise de l'outil informatique en planétologie

Partie FORTRAN

Aymeric SPIGA

December 1, 2019

Cours 1 : Initiation au langage Fortran (et un peu d'Unix)

1. Informatique et programmation
2. UNIX/Linux
3. FORTRAN
4. Debugging (slides par J.-B. Madeleine)

Plan

L'informatique : un outil puissant...

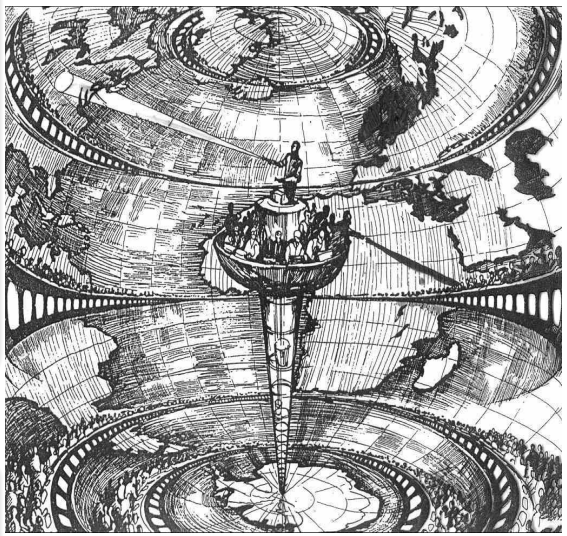
En islandais $\Rightarrow$ *tölva* [ordinateur]
$=$ *völva* (magicienne) $+$ *tala* (chiffre)

Omniprésence de la programmation

- ✎ On comprend en modélisant;
- ✎ Donner vie à des systèmes complexes dont la description analytique serait titanesque (rêve de Richardson);

"Langages" de programmation

- ☞ Apprentissage abstrait ($\sim$ grammaire)
- ☞ Apprentissage pratique ($\sim$ acheter à manger, vie sociale, ...)



[L'usine météorologique de Lewis Fry Richardson]

. . . et délicat à maîtriser

État d'esprit à adopter

- ☞ Forger pour devenir forgeron (se lancer et faire des erreurs).
- ☞ Adopter une approche modulaire (construction d'une maison).
- ☞ Toujours penser à la pérennité du code (évolution).
- ☞ Être méticuleux et patient.
- ☞ Lire le manuel (auto-formation permanente).

Principes de base

- ☞ Un bon code est simple, bien organisé et commenté.
- ☞ Le programme fait simplement ce qu'on lui demande.
- ☞ Un ordinateur ne peut faire un calcul exact.
- ☞ Loi de Murphy et probabilité du *bug*.

Plan

Systèmes d'exploitation UNIX et GNU/Linux

- ☞ Au coeur de iOS et Android ($\sim$ 85% des smartphones)
- ☞ Utilisé sur 95% des 500 supercalculateurs les plus puissants
- ☞ Beaucoup utilisé dans les laboratoires scientifiques

Interface entre vous et le système d'exploitation Linux appelé **"shell"** (" coquille" du système), accessible par un **terminal** (ou console).

Commandes usuelles du “shell” UNIX

Syntaxe générale des commandes :
`commande [options] arguments`

Commande	Effet	Exemple
`pwd`	indiquer le répertoire courant	`pwd`
`ls`	lister le contenu d'un répertoire	`ls -l *.F90`
`cd`	changer de répertoire	`cd /media/cle`
`cp`	copier un fichier	`cp t.F90 t_sav.F90`
`mv`	renommer un fichier	`mv t.F90 tt.F90`
`rm`	supprimer un fichier	`rm pourri.F90`
`mkdir`	créer un répertoire	`mkdir minipro`
`top`	regarder les processus actifs	`top`
`man`	informations sur une commande	`man ls`
`cat`	explorer un fichier sans défilement	`cat matinput.txt`
`more`	explorer un fichier avec défilement	`more dat.txt`

Bon à savoir: `UNIX`

- ☞ Chemin absolu d'un répertoire `/home/tartenpion/cle_usb`
- ☞ Chemin relatif d'un répertoire `../../document`
- ☞ Pour arrêter une commande bloquée au premier plan `Ctrl+C`
- ☞ Pour arrêter une commande bloquée en arrière plan, `top` puis `k` et entrer process ID, sortir avec `q`
- ☞ Pour indiquer tous les caractères possibles en nombre illimité `*`
- ☞ Pour rediriger la sortie d'un programe
  `ls > fichier.txt 2> erreurs.txt`
- ☞ Pour indiquer des paramètres d'entrée d'un fichier
  `prog.exe < fichier.txt`

Plan

FORTRAN (FORmula TRANslator)

- Créé en 1954 (préhistoire de l'informatique)
- Langage de haut niveau (non compilé)
- Puissance et efficacité en calcul scientifique (astronomie, climat, météorologie, aérospatiale, aéronautique, dynamique des fluides, modélisation des réactions chimiques, modèles économiques...)
- Fortran 90 : tableaux multi-dimensionnels, modularité, ...
- Intuitif, mais demande d'être très précis lors du codage
- Compilateurs gratuits (`gfortran`) ou payants (`pgf`, `ifort`)
- Surlignage syntaxique dans la plupart des éditeurs de texte

Méthode de travail rigoureuse en quatre temps

Conception

avec un papier et un crayon (et un cerveau: numérisation des équations, organigrammes de programmation)

Codage

avec un clavier et un éditeur de texte

Compilation

pour générer le programme qui exécute le code source

Exécution

données d'entrée, paramètres, et esprit critique face aux résultats

Syntaxe du FORTRAN

prog/bonjour.F90

```
! Exemple simple de programme
! Auteur: M. Planeto
! Date: 03/03/2013
PROGRAM bonjour
IMPLICIT NONE ! Cette ligne est necessaire
    write(*,*) "Bonjour"
    write(*,*) &
      "Mon␣cher␣ami"
END PROGRAM bonjour
```

- ☞ Pas de distinction majuscules vs. minuscules
- ☞ Indifférence au nombre d'espaces et à l'indentation
- ☞ Un texte qui suit un ! est en commentaire
- ☞ Une esperluette & en fin de ligne prolonge à la ligne suivante
- ☞ Toute boucle ou procédure ouverte doit être fermée

Compilation d'un programme FORTRAN

Il faut recompiler à chaque fois que l'on modifie le code source

A partir d'un code source `toto.F90` écrit en langage `FORTRAN` dans un éditeur de texte, on crée un exécutable `toto.exe` interprétable par l'ordinateur. On utilise le compilateur libre `gfortran` avec l'option `-Wall` pour avoir tous les avertissements.

Compilation programme simple

```
gfortran -Wall toto.F90 -o toto.exe
```

Compilation programme + module(s)

```
gfortran -Wall module.F90 toto.F90 -o toto.exe
```

Exemples

- ☞ `gfortran -Wall bonjour.F90 -o bonjour.exe`
- ☞ `gfortran -Wall operations.F90 pi.F90 -o pi.exe`

Déclaration des variables : passage obligé

prog/declaration.F90

```
PROGRAM declaration
IMPLICIT NONE
 ! IL FAUT COMMENCER PAR DECLARER LES VARIABLES
 ! ... afin de reserver une zone memoire
 integer :: i              ! un entier court sur 32 bits
 integer*8 :: j            ! un entier long sur 64 bits
 real :: r                 ! un reel sur 32 bits
 double precision :: d     ! un reel sur 64 bits
 character(len=3) :: c     ! une chaine de 3 caracteres
 logical :: is_it          ! un booleen
 integer :: k = 2                  ! un entier initialise
 real,parameter :: grav = 3.72     ! un reel constant
 ! ENSUITE ON PEUT DETAILLER LES INSTRUCTIONS
 i = 2             ! ici on affecte une valeur
 is_it = .true.    ! ici on affecte une valeur
 r = real(i)       ! ici on convertit un entier en reel
 i = int(grav)     ! ici on prend la partie entiere
END PROGRAM declaration
```

Note : entier 32 bits $<= 2^{31} - 1$; entier sur 64 bits $<= 2^{64} - 1$

Opérations sur des variables

prog/operation.F90

```
PROGRAM operation
IMPLICIT NONE
 ! On declare les variables...
 integer :: i=-2, j=4, k
 real :: pi = 3.141, res
 ! ... ensuite seulement on peut donner des instructions
 k = i+j     ! addition
 k = i-j     ! soustraction
 k = i*j     ! multiplication
 k = i/j     ! division
 k = i**j    ! puissance
 ! on peut aussi appliquer des fonctions
 ! ... presentes par defaut (intrinseques)
 res = cos(pi)
 k = abs(i)
END PROGRAM operation
```

Booléens et comparaisons

Qu'imprime le programme en chacune des lignes ?

prog/booleen.F90

```
PROGRAM booleen
IMPLICIT NONE
 integer :: i=2, j=4
 logical :: cond
 ! exemple de comparaisons
 cond = (i == j)      ; print *, cond
 cond = (i /= j)      ; print *, cond
 cond = (i > j)       ; print *, cond
 cond = (i < j)       ; print *, cond
 cond = (i >= j)      ; print *, cond
 cond = (i <= j)      ; print *, cond
 ! exemple d'operations sur booleen
 cond = .NOT.(i <= j)              ; print *, cond
 cond = (i == j) .AND. (i /= j) ; print *, cond
 cond = (i == j) .OR. (i /= j)  ; print *, cond
END PROGRAM booleen
```

Procédures et approche modulaire

PROGRAM (∼ atelier)

Le programme principal. Idéalement extrêmement court. Fait appel à des SUBROUTINE ou des FUNCTION contenues dans un ou des MODULE.

MODULE (∼ boîtes à outils)

Un ensemble qui contient des SUBROUTINE ou des FUNCTION.

SUBROUTINE (∼ outil)

Un sous-programme qui opère sur une partie des arguments d'entrée.

FUNCTION (∼ outil)

Une fonction qui calcule un résultat d'après les arguments d'entrée.

Construction : MODULE, SUBROUTINE, FUNCTION

prog/imprime.F90

```
! Un module pour manipuler et imprimer des nombres
MODULE imprime
IMPLICIT NONE
CONTAINS

 SUBROUTINE invite(i) ! Demande un nombre a l'utilisateur
   integer :: i
   write(*,*) "Cher utilisateur, veuillez saisir un entier"
   read(*,*) i
 END SUBROUTINE invite

 SUBROUTINE carre(nombre,nombrecar) ! Calcule le carre
   integer :: nombre, nombrecar
   nombrecar = nombre * nombre ! (attention si on modifie nombre)
 END SUBROUTINE carre

 integer FUNCTION suivant(nombre) ! Renvoie le nombre suivant
  integer :: nombre
  suivant = nombre + 1
 END FUNCTION suivant

END MODULE imprime
```

Construction : PROGRAM

prog/chiffre.F90

```
! Exemple simple de programme modulaire
! Auteur: M. Planeto 03/03/2013
PROGRAM chiffre
USE imprime ! On importe le contenu de imprime
IMPLICIT NONE
  integer :: i, i2
  CALL invite(i)
  CALL carre(i,i2)
  i = suivant(i2)
  CALL carre(i,i2)
END PROGRAM chiffre
```

- ☞ Organisation optimale et concise
- ☞ Compréhension aisée
- ☞ Portabilité
- ☞ Tâches répétitives déléguées

Autre exemple de MODULE : constantes planétaires

prog/mars.F90

```
! constantes planetaires de Mars
MODULE mars
IMPLICIT NONE
   real, parameter :: grav = 3.72       ! gravite en m/s2
   real, parameter :: albedo = 0.25     ! albedo
   real, parameter :: solarflux = 589. ! flux solaire en W/m2
END MODULE mars
```

prog/t_equivalent.F90

```
! programme pour calculer t_equivalent. M. Planeto. 03/03/2013
PROGRAM t_equivalent
USE mars
IMPLICIT NONE
   real, parameter :: sigma = 5.67e-8 ! Stefan-Boltzmann
   real :: t_eq
   t_eq = solarflux * (1.-albedo) / 4. / sigma
   t_eq = t_eq**0.25
   write (*,*) "Temperature equivalente de Mars: ",t_eq
END PROGRAM t_equivalent
```

Algorithmique : boucles IF

prog/boucle_if.F90

```
! Exemple simple de boucle IF
! Auteur: M. Planeto 03/03/2013
PROGRAM boucle_if
USE imprime
IMPLICIT NONE
  integer :: i, j
  CALL invite(i)
  CALL invite(j)
  IF (i == j) THEN                !! SI CONDITION VRAIE...
    write(*,*) "Nombres egaux"    !! ...ACTION
  ENDIF                           !! FERMETURE DE BOUCLE
END PROGRAM boucle_if
```

NB: Remarquer l'indentation des boucles pour plus de clarté.

Algorithmique : boucles IF...ELSE

prog/boucle_if2.F90

```
! Exemple de boucle IF ... ELSE
! Auteur: M. Planeto 03/03/2013
PROGRAM boucle_if2
USE imprime
IMPLICIT NONE
  integer :: i, j
  CALL invite(i)
  CALL invite(j)
  IF (i == j) THEN                      !! SI CONDITION VRAIE...
    write(*,*) "nombres egaux"          !! ... ACTION
  ELSE                                  !! SINON...
    write(*,*) "nombres differents"     !! ... ACTION
  ENDIF                                 !! FERMETURE DE BOUCLE
END PROGRAM boucle_if2
```

Algorithmique : boucles IF...ELSE IF...ELSE

prog/boucle_if3.F90

```
! Exemple de boucle IF ... ELSE IF ... ELSE
! Auteur: M. Planeto 03/03/2013
PROGRAM boucle_if3
USE imprime
IMPLICIT NONE
  integer :: i, j
  write (*,*) "ATTENTION␣JE␣VEUX␣DEUX␣NOMBRES␣DIFFERENTS"
  CALL invite(i)
  CALL invite(j)
  IF (i > j) THEN                    !! SI CONDITION 1 VRAIE...
    print *,"nombre␣1␣superieur"     !! ... ACTION
  ELSE IF (i < j) THEN               !! SINON SI CONDITION 2 VRAIE...
    print *, "nombre␣1␣inferieur"    !! ... ACTION
  ELSE                               !! SINON
    print *,"et␣la␣consigne␣?"       !! ... ACTION
  ENDIF                              !! FERMETURE DE BOUCLE
END PROGRAM boucle_if3
```

Algorithmique : boucles DO

prog/boucle_do.F90

```
! Exemple de boucle DO
! Auteur: M. Planeto 03/03/2013
PROGRAM boucle_do
USE imprime
IMPLICIT NONE
  integer :: i, i2, nfin
  CALL invite(nfin)
  DO i=0,nfin,2            !! POUR CHAQUE i dans [0,nfin] a pas de 2
    CALL carre(i,i2)       !! ... ACTION dependant de i
    print*,i,"carre=",i2   !! ... ACTION dependant de i
  END DO                   !! FERMETURE DE BOUCLE
END PROGRAM boucle_do
```

NB: Remarquer à nouveau la puissance de l'approche modulaire.

Algorithmique : boucles WHILE

prog/boucle_while.F90

```
! Exemple de boucle WHILE
! Auteur: M. Planeto 03/03/2013
PROGRAM boucle_while
USE imprime
IMPLICIT NONE
  integer :: i, i2, nfin
  CALL invite(nfin)
  i = 0
  DO WHILE (i < nfin)     !! TANT QUE CONDITION VRAIE...
    CALL carre(i,i2)      !! ... ACTION
    print*,i,"carre=",i2  !! ... ACTION
    i = i+1               !! ... ACTION necessaire ici
  END DO                  !! FERMETURE DE BOUCLE
END PROGRAM boucle_while
```

Qu'est-ce qui ne va pas avec ce programme ?

7 erreurs ou maladresses à trouver

prog/comparaison_bad.F90

```
PROGRAM comparaison
integer :: i
write(*,*) "saisir␣deux␣entiers"
read(*,*) i, j
if (i = j) then
write(*,*) "egaux"
end if
write(*,*) "saisir␣deux␣entiers"
read(*,*) i, j
if (i > j) then
write(*,*) "premier␣>␣second"
end if
END PROGRAM affectation
```

Un exemple récapitulatif: calcul de π

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} \cdots$$

prog/operations.F90

```
! operations: un module qui contient des operations utiles
! que l'on fait tout le temps. Auteur: M. Planeto. 03/03/2013
MODULE operations
IMPLICIT NONE
CONTAINS
  integer*8 FUNCTION chgtsigne(i)
    integer*8 :: i
    chgtsigne = -i
  END FUNCTION chgtsigne

  double precision FUNCTION divide(i,j)
    integer*8 :: i,j
    ! sans les dble(), on ferait la division euclidienne ...
    divide = dble(i) / dble(j)
  END FUNCTION divide
END MODULE operations
```

Un exemple récapitulatif: calcul de π

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} \cdots$$

prog/pi.F90

```
! Programme qui calcule pi
! par la formule pi/4 = 1 - 1/3 + 1/5 - 1/7 ...
! Auteur: M. Planeto. 03/03/2013
PROGRAM pi
USE operations
IMPLICIT NONE
   double precision :: s=4.
   integer*8 :: i=1,j=1,iter=0,nfin
   write(*,*) "Il␣me␣faut␣le␣nombre␣d'iterations"
   read(*,*) nfin
   DO WHILE (iter <= nfin)
      i = chgtsigne(i)
      j = j + 2
      s = s + 4.d0*divide(i,j)
      iter = iter + 1
      write(*,*) iter,s
   ENDDO
END PROGRAM pi
```

Écrire les résultats dans un fichier texte

prog/writefile.F90

```
! Ecrire dans un fichier. Auteur: M. Planeto. 03/03/2013
PROGRAM writefile
USE imprime
IMPLICIT NONE
   integer :: i=1,nfin,s=1
   integer :: ref_file
   CALL invite(nfin)
   ref_file = 1                              ! DEFINIR REF
   OPEN(UNIT=ref_file,FILE="dat.txt")        ! OUVRIR FICHIER
   DO WHILE (i<nfin)
      i = i + 2
      s = s + i
      WRITE(ref_file,*) i,s                  ! ECRIRE FICHIER
   ENDDO
   CLOSE(ref_file)                           ! FERMER FICHIER
END PROGRAM writefile
```

Manipulation de tableaux multidimensionnels

prog/tab.F90

```
! Tableaux multidimensionnels. Auteur: M. Planeto. 03/03/2013
PROGRAM tab
IMPLICIT NONE
   real, dimension(3,3) :: tableau     ! matrice 3x3
   real, dimension(3) :: vecteur, res ! vecteurs
   integer :: i
   vecteur = (/ 1, 2, 3 /)  ! Definir un vecteur "a la main"
   OPEN(UNIT=98,FILE="matinput.txt")
   DO i=1,3
     READ(98,*) tableau(i,:) ! Lire une matrice ligne a ligne
   ENDDO
   CLOSE(98)
   print *, SIZE(tableau) ! Autres: SHAPE, MAXVAL, MINLOC, etc...
   tableau(:,:) = cos(3.14*tableau(:,:)/6.)  ! Operations
   WHERE (tableau > 0)                ! Boucle WHERE
       tableau = - tableau
   ELSEWHERE
       tableau = tableau**2
   END WHERE
   res = MATMUL(transpose(tableau),vecteur)  ! Usage avance
   print *, res
END PROGRAM tab
```

Plan

Debugging

Tout un art...

- ☞ Bug peut venir d'une erreur de calcul qui se propage: savoir remonter l'erreur avec des `print` et resserrer les mailles (image du loup dans la forêt)
- ☞ Bug peut être aléatoire: comportement différent selon la machine, le compilateur (pire bug, peut rendre fou)

...mais des outils existent

- ☞ Options du compilateur pour détecter les erreurs (pour gfortran par exemple: `-g -Wall -fbounds-check -ffpe-trap=invalid,zero,overflow`)
- ☞ Outils de profiling (par exemple gprof)
- ☞ Exécution pas à pas (par exemple TotalView)

...en tout cas, une règle d'or: si vous saturez, sortez prendre l'air!

Exemple de debugging 1/3

Objectif du programme

Soit la function $f(x) = \sqrt{x^2+3x} - \sqrt{x^2+2x}$; Que vaut $\lim_{x\to\infty} f(x)$? (réponse analytique: $1/2$)

Méthode brute

- ☞ on code la fonction
- ☞ on fait tendre x vers un nombre très grand avec une boucle
- ☞ on ne teste pas le comportement aux limites
- ☞ on croit que tout fonctionne bien

Exemple de debugging 2/3

prog/simplelimite.F90

```
PROGRAM simplelimite
! Author: J-B Madeleine, based on a class by J. Chang (NASA Ames)
! Estimate limit as X approaches infinity; notice:
!   - the effect of forgetting the IMPLICIT NONE
!   - the difference in two writings of the same equation
!   - the difference between SINGLE and DOUBLE precision
!IMPLICIT NONE
INTEGER :: i
!REAL :: x, y1, y2
!DOUBLE PRECISION :: x, y1, y2

x = 1.0d0
!do i = 1,30
do i = 1,10
  y1 = sqrt(x*x + 3.0d0*x) - sqrt(x*x + 2.d0*x)
!  y2 = 1 / (sqrt(1.d0 + 3.0d0/x) + sqrt(1.d0 + 2.d0/x) )
!  print *, 'x,y1,y2 = ',x,y1,y2
!  print *, 'x,y1 = ',x,y1
  x = x*10.0d0
enddo
print *, 'x,y1 = ',x,y1
END
```

Exemple de debugging 3/3

Bonne méthode

- ☞ on vérifie le comportement aux limites: `NaN`
- ☞ on compile avec plus d'options pour en savoir plus (compilation+exécution): on obtient zéro en double précision !
- ☞ on réfléchit:

$$f(x) = \frac{\left(\sqrt{x^2+3x} - \sqrt{x^2+2x}\right)\left(\sqrt{x^2+3x} + \sqrt{x^2+2x}\right)}{\left(\sqrt{x^2+3x} + \sqrt{x^2+2x}\right)}$$

$$f(x) = \frac{x}{\left(\sqrt{x^2+3x} + \sqrt{x^2+2x}\right)} = \frac{1}{\left(\sqrt{1+3/x} + \sqrt{1+2/x}\right)}$$

- ☞ on prend cette écriture, qui évite de calculer $\infty - \infty$; on obtient un code robuste, même en simple précision.