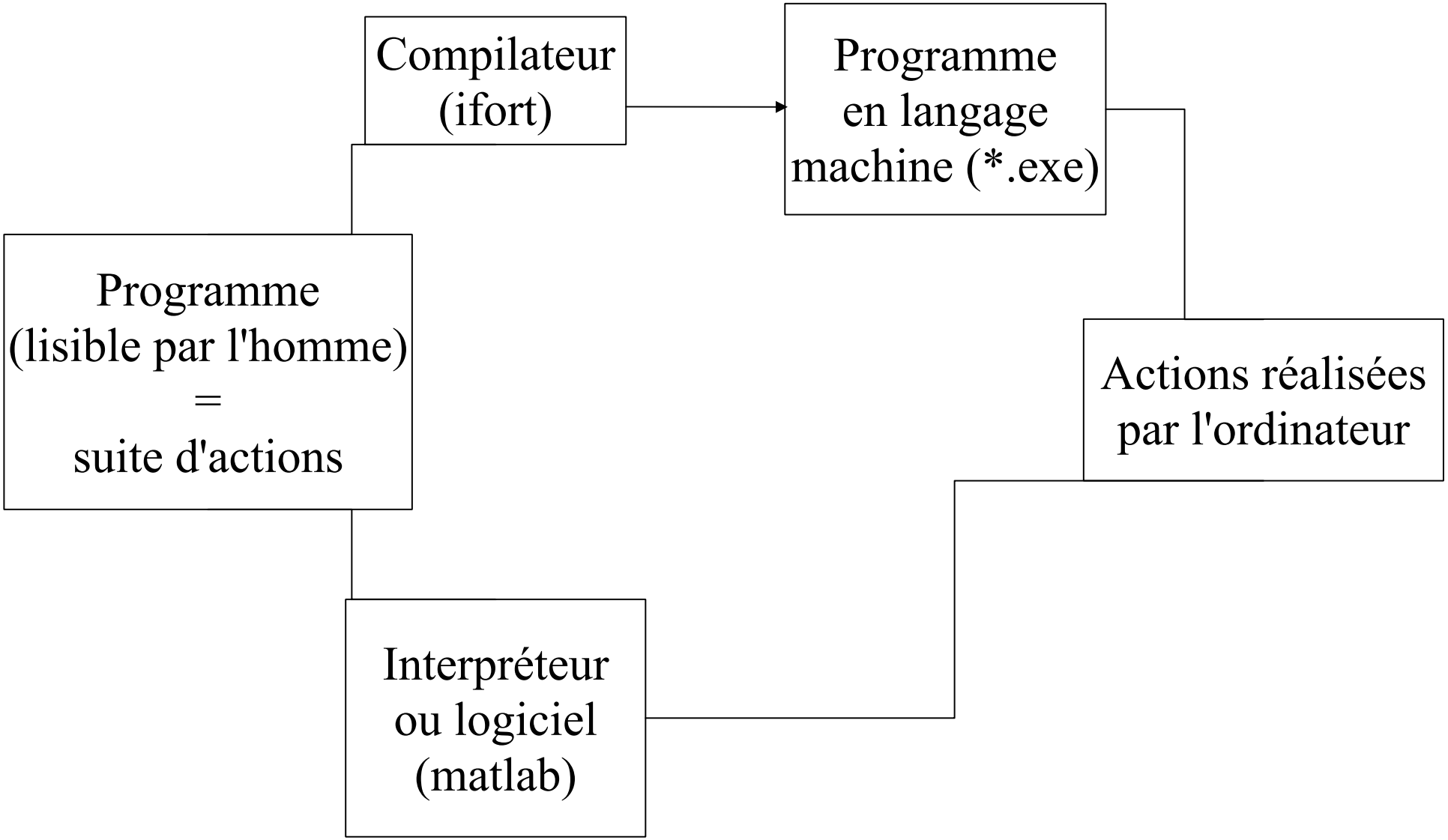
Cours Informatique Master STEP

- Bases de la programmation:
  - Compilateurs/logiciels
  - Algorithmique et structure d'un programme
- Programmation en langage structuré (Fortran 90)
  - Variables, expressions, instructions simples
  - Instructions complexes (boucles, tests...)
  - Tableaux et variables structurées
  - Sous-programmes et fonctions
- www.ipgp.jussieu.fr/~narteau/DATA/cours_info.pdf
- www.idris.fr/data/cours/lang/fortran/fortran_base/IDRIS_Fortran_cours.pdf
- www.mip.ups-tlse.fr/~mieussens/PAGE_WEB/ENSEIGNEMENT/cours_f90.pdf

Qu'es-ce qu'un programme



Méthode de programmation

- But du programme?
- Variables d'entrée (données)?
- Variables de sortie (produits du programme)?
- Variables internes?
- Décomposer les actions à faire:
  - Lire les entrées
  - Calculs/actions à faire
  - Ecriture des sorties
- Ecrire le programme en langage informatique

- position d'un satellite f(t)
- Paramètres de l'orbite,

  temps t
- (r, Θ, Φ) (t)
- Masse de la planète, constante de gravitation...
- Lire le temps demandé
- calcul de la position
- écrire la position avec un certain fomat (Vecteur ou coordonnées sphériques)

Exemple de programme

Structure d'un programme

- Structure ci-contre
- Mettre des commentaires (!) pour décrire les variables et les instructions
- Assurer la lisibilité du programme:
  - Sauter des lignes
  - indenter (marge) les différentes parties
- Valide pour tout langage

```
! Commentaire décrivant le but, le contenu,
! l'auteur, la date, etc du programme

PROGRAM [nom_du_programme]

[ Déclaration:
[ des variables (entrées, sorties, locales),
[ des fonctions et des sous-programmes

[ Corps du programme:
[ Instructions, structures et actions
[ décomposées en actions simples

END PROGRAM
```

Créer, compiler et exécuter un programme en fortran90

- Rédiger son programme au format ascii dans un éditeur de texte quelconque (xemacs préféré).
- Sauver son fichier avec l'extension .f90 (toto.f90)
- Compiler le programme par la ligne de commande:

>ifort -o toto.exe toto.f90

- Si aucune érreur, exécuter le programme par:

>toto.exe

- Cette suite de commande peut etre mise dans un script

Programmer en Fortran 90
Qu'es-ce qu'une variable?

- Une variable est l'unité qui va contenir l'information dans un programme
- Une variable possède:
  - un nom de variable -> adresse de l'espace mémoire réservé
  - un type (réel, entier, complexe, booléen, chaine de caractères...)
  - un espace mémoire réservé lors de sa déclaration

Variables entières (INTEGER)

- Déclaration:

INTEGER :: [nom_variable] ! i, j, k !

- Expressions:

i=j+k , i=j*k , i=j/k

- Fonctions simples:
  - x=real(i) : renvoie le réel x associé à l'entier i
  - mod(i,40) : renvoie le modulo de i par 40

Variables réelles (REAL)

- Déclaration:

REAL :: [nom_variable] ! x, y, z !

- Expressions:

x=y+z , x=y*z , x=y/z

- Fonctions simples:
  - i=int(x) : renvoie la partie entière de x dans l'entier i
  - y=cos(x), y= exp(x) ....

Variables complexes (COMPLEX)

- Déclaration:

COMPLEX :: [nom_variable] ! X, Y, Z !

! contient deux réels pour les parties réelles et imaginaires

- Expressions: X=Y+Z , X=Y*Z
- Fonctions simples:
  - x=real(X) : renvoie la partie réelle de X dans le réel X
  - y=Im(X) : renvoie la partie imaginaire de X dans le réel y
  - Y= exp(X) ....

Variables Booléennes (LOGICAL)

- Déclaration:

LOGICAL :: [nom_variable] ! I, J, K !

! Ne prend que deux valeurs: TRUE (1) ou FALSE (0)

- Expressions:
  - I=J.AND.K , I=J.OR.K, I=.NOT.J (tableau des relations...)
  - I=(i<j) , I=(x>y) , I=(x>=y) , I=(i<=j) , I=(i==j) , I=(i/=j)
  - I=((x<y).AND.(x>z)) , I=((x>z).OR.(x>y))

Variables caractères (CHARACTER)

- Déclaration:

CHARACTER(30) :: [nom_variable] ! A,B !

! Le chiffre entre parenthèses indique le nombre de caractères dans la chaine. Si on ne met pas le chiffre, alors 1 caractère

- Expressions:
  - I=(A==B) renvoie TRUE uniquement si les chaines de caractères sont identiques dans les variables A et B
- Fonctions simples:
  - i=len(A) renvoie le nombre de caractères dans A
  - i=len_trim(A) renvoie le nombre de caractères non blancs

Instructions simples d'entrée/sortie

- WRITE(*,*) 'J'ai ', i, ' ans'

! Ecris à l'écran la chaine de caractères entre les cotes ('), la valeur de la variable i, puis l'autre chaine de caractères

- READ(*,*) x

! Lit au clavier le nombre donné, et le stocke dans la variable x

- (*,*) -> (unité d'entrée/sortie,format d'entrée/sortie)

Instructions complexes

- Condition (IF ... THEN ... ELSE ... END IF)
- Boucle itérative (DO ... END DO)
- Boucle Conditionelle (DO WHILE ... END DO)
- Instructions liées aux boucles (EXIT, CYCLE)
- Choix discret (SELECT CASE ... END SELECT)

Instructions complexes: Condition

```
IF ((x<0).OR.(x=0)) THEN  ! condition
   x=0.0-x                                  ! si condition
   WRITE(*,*) 'x était négatif ou nul'   ! si TRUE
ELSE
   WRITE(*,*) 'x est positif'               ! si FALSE
END IF
```

- Le ELSE est facultatif (mais pas le END IF)

Instructions complexes:
Boucle itérative

```
j=1
DO i=0,100,2  ! le compteur est i (ne pas modifier
  ! dans la boucle) i=0 au premier tour,
  ! puis est incrémenté de 2 jusqu'à 100
   j=j*(i+1)*(i+2)
END DO ! renvoie au début de la boucle, incrémente i
WRITE(*,*) 'factorielle ... égale ', j
```

Instructions complexes:
Boucle conditionelle

```
j=1
DO WHILE (i<100) ! condition pour entrer
   i=i+2
   j=j*(i+1)*(i+2)
END DO ! renvoie à la condition du début
WRITE(*,*) 'factorielle ... égale ', j
```

Instructions liées aux boucles

- EXIT

! sort de la boucle lorsque le programme recontre EXIT

- CYCLE

! va jusqu'au prochain END DO sans effectuer

! les opérations entre CYCLE et END DO

Instructions complexes
Choix discret

```
READ(*,*) REPONSE
SELECT CASE (REPONSE)
   CASE ('y')                          ! si REPONSE='y'
     WRITE(*,*) 'Réponse YES'
  CASE ('n')                           ! si REPONSE='n'
     WRITE(*,*) 'Réponse NO'
  CASE DEFAULT                         ! si aucun des autres cas
     WRITE(*,*) 'Vous avez répondu ni YES, ni NO'
END SELECT
```

Les Tableaux

- On peux organiser une suite de variables d'un meme type dans un tableau.
- Le Tableau est caractérisé par:
  - un nom de tableau
  - un type de variables stockées dans le tableau
  - le nombre de dimensions du tableau (1D = vecteur, 2D = matrice ...)
  - la longueur du tableau selon chaque dimension

Déclaration des tableaux

- on commence par le type des variables
- puis la longeur selon chaque dimension
- enfin le nom du tableau
- éventuellement son initialisation (remplissage) par une valeur

Exemples:

- REAL, DIMENSION(3) :: U, V, W
- CHARACTER(30), DIMENSION(2) :: NOM
- REAL, DIMENSION(3,3) :: MATRICE=0.0

Accéder aux éléments du tableaux

- Chaque élément du tableau est répéré par un nombre d'indices entiers qui dépend de la dimension du tableau
- Ces indices vont de la valeur 1 à la longueur du tableau dans cette dimension

Exemple:

```
DO i=1,3
    x=V(i)*U(i)                                  ! produit scalaire U.V
    DO j=1,3
        W(j)= W(j) + MATRICE(i,j)*V(j) ! produit matrice par V
    END DO
END DO
```

Erreurs à éviter avec les tableaux

- Ne pas confondre les indices (i,j) qui servent à repérer un élément du tableau avec la valeur contenue dans cet élément du tableau:

WRITE(*,*) i,j différent de WRITE(*,*) MATRICE(i,j)

- Ne pas donner un indice i dont la valeur ne serait pas comprise entre 1 et la dimension du tableau -> renvoie n'importe quelle valeur

WRITE(*,*) V(4), MATRICE(5,6) ! donne n'importe quoi

- Manipuler avec précaution les opérations prédéfinies sur les tableaux où ceux-ci apparaissent sans leurs indices:

V=U/3.0 ! divise tous les éléments du tableau par la valeur 3.0

WRITE(*,*) MATRICE ! écrit à l'écran tous les membres du tableau

Expressions spécifiques aux tableaux

- On peux faire des opérations sur certains éléments du tableaux en utilisant des résumés d'indice:

```
WRITE(*,*) MATRICE(1,:) ! écris la première ligne de MATRICE
V(1:3)=-1.0 ! Donne la valeur -1.0 aux éléments dont les indices sont entre 1 et 3
```

- Si, et **uniquement si**, les **dimensions s'accordent en nombre et en taille**, on peux faire des opérations sur les tableaux du type:

```
U=MATRICE(:,1)*V(:)    ! U(i)=MATRICE(i,1)*V(i)
```

Instruction spécifique aux tableaux

- L'instruction WHERE permet de rechercher dans un tableau:

```
WHERE (MATRICE<2.0)
! cherche les indices qui vérifient la condition
    MATRICE=0.0 ! modifie uniquement les valeurs à ces indices
ELSEWHERE
    MATRICE=MATRICE*2.0 ! modifie les autres valeurs
END WHERE
```

- Comme pour IF, le ELSEWHERE est facultatif (mais le END WHERE)

Fonctions spécifiques aux tableaux (voir polycopié)

- MATRICE2=TRANSPOSE(MATRICE)
- W=DOT_PRODUCT(U,V)
- U=MATMUL(MATRICE,V)
- i=MINLOC(V) ! renvoie la valeur de l'indice dont l'élément du tableau est minimum
- j=MAXLOC(V) ! renvoie la valeur de l'indice dont l'élément du tableau est minimum
- .....

Fonctions et sous-programmes

- La réalisation d'un programme requiert de décomposer les actions en éléments de base, à la fois pour une meilleure compréhension du programme et pour pouvoir utiliser ces éléments de base dans d'autres programmes.
- Une fonction ou un sous-programme est **un petit programme indépendant**, avec entrées et sorties, qui sera écrit dans le meme fichier que la programme (entre la commande CONTAINS et END PROGRAM) ou dans un fichier différent qui sera apellé au moment de la compilation.

Fonctions intrinsèques (pré-définies)

- Chaque langage possède des fonctions intrinsèques que l'on peux utiliser dans nos programmes:

y=sin(x) ! entrée = réel x, sortie = sin(x)

x=real(X) ! entrée = complèxe X, sortie = partie réelle

i=MINLOC(V) ! entrée = tableau V, sortie = indice du min.

- Ces fonctions sont listées dans le polycopié ou dans l'aide en ligne du fortran90
- Une fonction renvoie **une seule variable** en sortie

Programmer une fonction

- Une fonction se programme presque un programme
- Les différences sont:
  - Donner le type de la fonction (ou de la variable de sortie) au début de la fonction: REAL FUNCTION MAXTAB(T)
  - Déclarer explicitement les variables d'entrée par l'attribut INTENT(IN)
  - On utilise le nom de la fonction comme variable de sortie: MAXTAB=...
  - On termine par: END FUNCTION MAXTAB

Programmer une fonction : Exemple

```
REAL FUNCTION MAXTAB(T)
REAL, INTENT(IN), DIMENSION(:) :: T
INTEGER :: I
DO I=1,SIZE(T)
    IF (T(I)>MAXTAB) THEN
        MAXTAB=T(I)
    END IF
END DO
END FUNCTION MAXTAB
```

Sous-programmes

- Un sous-programme est aussi un petit programme indépendant qui s'insère entre les instructions CONTAINS et END PROGRAM dans le fichier du programme
- La différence avec les fonctions est qu'il peut renvoyer en sortie plus d'une variable
- On l'apelle dans le programme par l'instruction :

CALL [Nom_sous_programme]([variables_entrée_sortie])

- Il ne communique avec le programme qu'au travers des variables qui lui sont données. Le nom du sous-programme n'est pas une variable.

Sous-programmes intrinsèques

- Certains sous-programmes sont prédéfinis.
- Exemples:

```
CALL RANDOM_NUMBER(T)
! remplit le tableau T de réels tirés aléatoirement entre 0 et 1
CALL DATE_AND_TIME(date,time, zone,values)
! renvoie dans les variables la date, l'heure, ...
! au moment de l'éxecution du sous-programme
```

Programmer un sous-programme

- Un sous-programme ressemble a un programme.
- Les différences sont:
  - Commencer par déclarer le sous-programme avec ses variables d'entrée, et de sortie (ordre important):

    SUBROUTINE MUL23(I,J,K)

  - Déclarer les variables d'entrée par INTENT(IN) et de sortie par INTENT(OUT).
  - Terminer par:

    END SUBROUTINE MUL23

Programmer un sous-programme Exemple

```
SUBROUTINE MUL23(I,J,K)
INTEGER, INTENT(IN) :: I
INTEGER, INTENT(OUT) :: J,K
J=2*I
K=3*I
END SUBROUTINE MUL23
```

- On l'utilise dans le programme (plus haut) par l'instruction:

```
CALL MUL23(I,J,K)
! où I,J,K sont des variables déclarées du programme
```

Librairies de sous-programmes

- Il existe des sous-programmes déjà codés (et débuggés) en fortran90 pour faire de nombreuses choses (algèbre matricielle, décomposition de fonctions...)
- Dans la mesure du possible, il faut utiliser ces sous-programmes lorsque ils sont adaptés à vos problèmes car:
  - Ils n'ont pas de bug
  - Ils sont optimisés (temps de calcul faible):
    - spécifiques au processeur
    - permettent la paraléllisation du code à moindre cout par un changement de librairie (séquentiel->parallèle)

Comment utiliser une librairie de sous-programmes?

- Chercher un sous-programme adapté (aide en ligne, descriptif)
- Installer la librairie sur votre ordinateur (souvent déjà fait)
- Connaître et déclarer les entrées et sorties (types, dimensions, tailles...) du sous-programme pour l'utiliser en boite noire
- Coder l'appel au sous-programme (CALL ....)
- Lors de la compilation, apeller la librairie au moment de la compilation du programme (option -l en général):

```
>ifort -l lapack -o toto.exe toto.f90
```

- Certaines librairies sont déjà incluses dans certains compilateurs

Quelques librairies de sous-programmes

- BLAS et LAPACK (calcul matriciel, inversion de matrices...)
- IMSL et NAG (calcul mathématiques)
- Mais aussi quelques librairies spécialisées:
  - SPICE (NASA): librairie fortran pour les calculs d'orbite, d'orientation et de géométrie de objets planétaires
  - DXFortran, MPLOT: librairies graphiques
  - ....
- Quelques sites web pour trouver les bonnes librairies fortran:
  - www.netlib.org et www.fortranlib.com