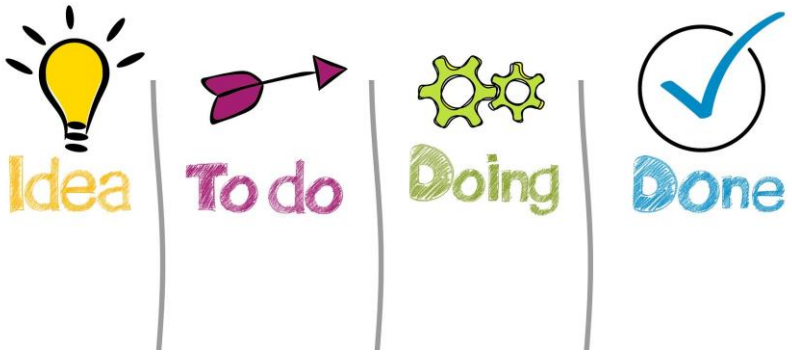
Éléments de chimie théorique ; HPC pour la chimie

CHPS1002

janvier 2025

Partie Fortran

Contact : Jean-Charles.Boisson@univ-reims.fr



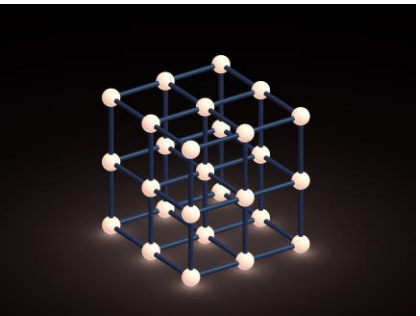
ORGANISATION

Image par Gerd Altmann de Pixabay

Image par Gerd Altmann de Pixabay

Organisation du module

- Double responsabilité :
  - Pr Éric Hénon (ICMR) :
    - acquisition de compétences/vocabulaire en chimie théorique;
    - utilisation de logiciel de visualisation.
  - Jean-Charles Boisson (LICIIS, LRC DIGIT) :
    - prise en main du langage Fortran (>=90) …
    - … dans un contexte HPC en chimie théorique (dont quantique).



Organisation du module

- Cours Magistraux (CMs) : 12h ⇔ 6 séances
  - ➔ Pr Hénon : 6h
  - ➔ Mr Boisson : 6h (☺)

Organisation du module

- Cours Magistraux (CMs) : 12h ⇔ 6 séances
  - ➔ Pr Hénon : 6h
  - ➔ Mr Boisson : 6h (☺)

- Travaux Pratiques (TPs) : 8h ⇔ 4 séances
  - ➔ Pr Hénon : 2h
  - ➔ Mr Boisson : 6h

Organisation du module

- Evaluation :
  - 1ère session :
    - CRTP : 40%
    - EET : 60%

Image par OpenClipart-Vectors de Pixabay

Organisation du module

- Evaluation :
  - 1ère session :
    - CRTP : 40%
    - EET : 60%
  - 2nde session :
    - CRTP : 40%
    - EOT : 60%

Image par OpenClipart-Vectors de Pixabay

FORTRAN

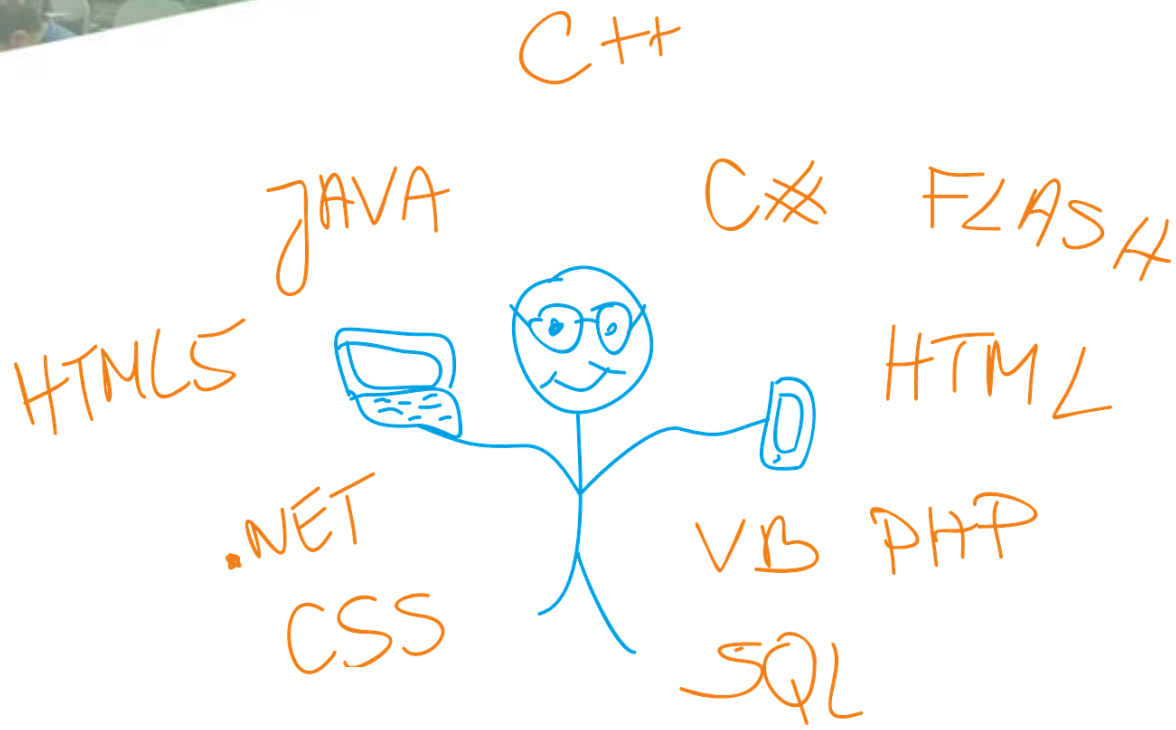
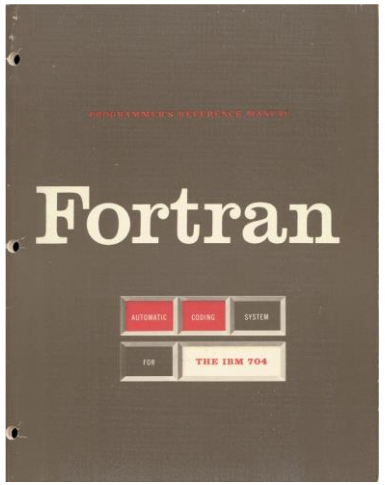


Image par kropekk_pl de Pixabay

Support de référence : « Programmer en Fortran », Claude Delannoy, éditions EYROLLES, ISBN 978-2-212-14020-0

Historique de Fortran

- FORTRAN ⇔ « FORmula TRANslator »
- Créé en 1954 …
- … compilé en 1956
- Dédié au calcul scientifique



15 octobre 1956
(image du domaine public)

Les versions de FORTRAN

- Normalisations :
  - FORTRAN ⇔ versions <= 77
    - Versions historiques : I, II, III et IV
      - Formatage lié aux cartes perforés (colonnes)
    - Versions 66 et 77 :
      - Ajout des instructions « classiques » : fonctions/procédures, types des données, primitives I/O, …

      ➔ Normalisation ANSI

Les versions de FORTRAN

- Normalisations :
  - Fortran ⇔ versions >= 90
    - 90 & 95 : formatage libre (☺), passage à la programmation modulaire, améliorations de la manipulation des tableaux, calcul parallèle, …
    - 2003 & 2008 : passage à la programmation orientée objet, enrichissement de la programmation parallèle, …
    - 2018 : mise à jour mineure de la version 2008

Fortran ?

- Est à la base d'un grand nombre de codes/bibliothèques en physique/chimie

- Est largement répandu dans le HPC
  - vectorisation
  - parallélisation :
    - OpenMP
    - MPI
    - GPU ⇔ CUDA Fortran

Fortran ?

- Flexibilité des tableaux comme Python
- POO comme JAVA
- Chaîne de compilation à la C/C++
- Utilisable directement via gfortran

A propos d'INTEL

- INTEL Parallel studio XE :
  - Prix : de 700 à 3000 $
  - Contient :
    - les versions optimisées INTEL :
      - des compilateurs C/C++/Fortran ⇔ icc / icpc / ifort
      - de Python 2 et 3
      - De l'implémentation MPI
    - Le profiler VTUNE
    - …

A propos d'INTEL

- INTEL Parallel studio XE :
  - Prix : de 700 à 3000 $
  - Contient :
    - les versions optimisées INTEL :
      - des compilateurs C/C++/Fortran ⇔ icc / icpc / ifort
      - de Python 2 et 3
      - De l'implémentation MPI
    - Le profiler VTUNE
    - …

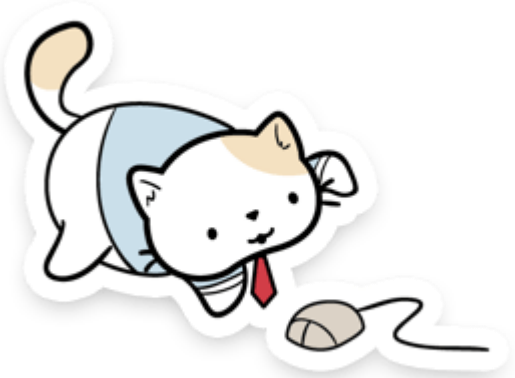
Gratuit pour les étudiants sous GNU/Linux

Remplacé par les oneAPI toolkits

A propos d'INTEL

- Depuis fin 2020 : arrivée des *Intel oneAPI toolkits*
- Equivalent au Parallel studio XE 2020 mais officiellement accessible librement
- Pour le HPC ⇔ 2 toolkits minimum
  - Intel oneAPI Base Toolkit
  - Intel oneAPI HPC Toolkit

PREMIERS PRINCIPES



Le format

- Depuis Fortran (>=90), le format n'est plus imposé ⇔ plus de notions :
  - de zone étiquette (colonnes 1 à 5)
  - de colonne suite (colonne 6)
  - De zone d'instructions (colonnes 7 à 72)

- On peut cependant encore écrire au format fixe
  ➔Ce n'est cependant pas conseillé

- Pas de casse à respecter

Le format

- Le symbole de commentaire est « ! »

  ➔ *integer :: i ! Déclaration d'un entier nommé i*

- Plusieurs instructions par ligne ⇔ utilisation du « ; »

  ➔ *integer :: i ; i = 2 ! Exemple pas très lisible …*

- instruction multi-lignes ⇔ utilisation du « & »

  ➔ *integer :: i, & ! Début de l'instruction*

  j ! Fin de l'instruction

Les types

- Entiers ⇔ *integer*
  - Écriture :
    - décimale : *+533 48 -2894*
    - en base :
      - 2 : *B'0110101110'*
      - 8 : *O'05472'*
      - 16 : *Z'B0FA'*

Les types

- Réels ⇔ *real*
  - Notation
    - décimale : *12.43 -0.38 -.38 4. .27*
    - exponentielle : *4.25E4 4.25e+4 5427e-34*
  - Version double précision :
    - Le type est : *double precision*
    - Notation exponentielle avec ‘d’ : *0.1d0*

Les types

- Le typage implicite est possible (*integer* ou *real*) selon le nom des variables

  ➔ procédé dangereux !!!

- Solution :
  - Utilisation de : *implicit none*
  - Empêche, par mégarde, le typage implicite

Les types

- Format des déclarations :

  ➔ *type :: nomVar [= value]*

- Quelques exemples :
  - *integer :: n = 6*
  - *real :: x = 1.5*
  - *double precision :: z = 5.25*
  - *integer :: i = 3, j = 10*
  - *logical :: x,y*
  - *logical :: z = 2<3*

Les types

- Déclaration de constantes :

  ➔ *type, parameter :: nomCste = value*

- Utilisable dans les autres déclarations

- Exemple :
  - *integer, parameter :: n = 100*
  - *Integer, parameter :: n2 = n**2*

Les types

- Booléens ⇔ *logical*
  - Valeurs :
    - false (faux) : *F ou .false.*
    - True (vrai) : *T ou .true.*
  - Opérateurs de comparaison (donnent des valeurs *logical*) :
    - <, <=, >, >=, ==, /=

      ➔ anciennement (F77) .LT., .LE., .GT., .GE., .EQ., .NE.
  - Composition avec les opérateurs :
    - .and., .or., .not. ⇔ et, ou (inclusif), non
    - .eqv. et .neqv. ⇔ au symbole logique ⇔ et sa négation

Les types

- Complexes ⇔ *complex*
  - Simple combinaison de deux valeurs réelles
  - Compatible avec les opérateurs arithmétiques usuels

  - Déclaration :

    complex :: z = (1, 2)

    ⇔ 1+2i
  - Accès aux composantes : *real(z)* et *aimag(z)*

Les types

- Chaînes de caractère ⇔ *character*
  - Pas de différenciation caractère unique et chaîne
  - Taille nécessaire à la déclaration :
    ➔ *character (20) :: mot ! Zone pouvant accueillir 20 caractères*
  - On peut aussi utiliser le mot clef « len » :
    ➔ *character (len=3) :: petitMot*
  - Rappel : la déclaration F77 est toujours possible

Les types

- Utilisation des *character* :
  - Si taille > contenu : complétion en espace
  - Si taille < contenu : troncature

  - Sous-chaînes ⇔ mécanisme comparable au slicing de Python (attention aux indices à partir de 1*)

    ➔ si mot vaut « bonjour »

    ➔ *mot(2:6)* vaudra « onjou »

Les types

- Utilisation des *character* :
  - La concaténation utilise le symbole « *//* »
  - Les espaces superflus s'enlèvent avec la fonction « *trim* »
  - Vraie taille d'une chaîne :
    - *len(trim(mot))* remplaçable directement par *len_trim(mot)*
  - Début d'une sous-chaîne ⇔ *index*

    ➔ *index (« bonjour », « on »)*

    ➔ réponse : 2

Les opérations arithmétiques

- Opérateurs classiques :
  - Binaires : +, -, /, *, **
    - / sur deux entiers donne un résultat entier
    - ** ⇔ mise à la puissance
    - Pas d'opérateur modulo % ⇔ fonction mod
  - Unaire : -

- Opérateurs sur les structures :

  ➔ (re)définition des opérateurs possible

Les structures de contrôle

- L'embranchement conditionnel ⇔ *if*

- Format condition simple :

  IF (test) then

  instructions

  END IF ! Ou ENDIF

- Si une seule instruction :

  IF (test) instruction

Les structures de contrôle

- Format attendu pour l'échec du test ⇔ « *else* »
- En cas de tests multiples ⇔ « *else if* »
- Avec Fortran est arrivé aussi le « *select case* »

Les structures de contrôle

- Utilisation du « select case » :

```
select case (valeur)
  case (valeurs ciblées)
     instructions
  case (valeurs ciblées)
     instructions
  case default
     instructions
```

Les structures de contrôle

- Les valeurs ciblées du *case* peuvent être :
  - Une seule valeur : *case(2)*
  - Plusieurs valeurs : *case(1,2)*
  - Un intervalle borné : *case(3:13)*
  - Un intervalle semi-borné : *case(:23)* ou *case(10:)*

Les structures de contrôle

- Cas des valeurs inattendues pour le select case
  ➔ passage dans le *default*

- Fonctionne aussi pour les types *character* ☺

```
select case (answer)
    case ('Yes')
        ...
    case ('No')
        ...
```

Les structures de contrôle

- La boucle au nombre d'itérations connu
  - Le « pour » en algorithmie

  - Utilisation du « do »

do var = begin, end, step ! step =1 si omis
instructions
end do

  - *do i = 1,10,2* ⇔ 1, 3, 5, 7 et 9

Les structures de contrôle

- La boucle au nombre d'itérations non connu
  - Le « tant que » en algorithmie

  - Utilisation du « do while»

do while(test)

instructions

end do

Les structures de contrôle

- Primitives de contrôles de boucle :
  - *exit* : permet de sortir directement d'une boucle

    ➔ utilisation avec la boucle infinie :

    ```
    do
      instructions
    end do
    ```

  - *cycle* : force le passage à l'itération suivante, le compteur est incrémenté mais les instructions suivant le cycle n'auront pas lieu

Nommage des structures de contrôle

- Permet de nommer le début de la déclaration des structures de contrôle ⇔ *nom :*

- Utile en cas de fortes imbrications pour savoir quand une structure se finit

  ⇔ exemple : *end do nom*

- Utile aussi avec l'utilisation du *cycle*.

Utilisation du GO TO

- Outil important en FORTRAN 77

- Permet de « brancher » directement à un endroit du code ⇔ une étiquette

  (nombre entier sans signe de 1 à 5 chiffres non nuls)

- Permet encore de gérer des cas exceptionnels

Utilisation du STOP

- Fin d'un programme en Fortran ⇔ mot clé *end*

- En cas d'un arrêt non classique :

  ➔Utilisation de *stop*

  ➔Suivi éventuellement d'une valeur numérique

  ⇔ similaire au code de sortie des scripts (exit)

Format d'un programme Fortran

- Extension de fichier ⇔ version du langage
- Pour nous, au minimum : .f90
- Format global d'un programme :

```
program nomProgramme
        instructions
end
```

LES ENTRÉES / SORTIES

Principe de lecture

- Primitive read :
  - Chaque appel ⇔ lecture d'une nouvelle ligne du flux
  - Séparateurs standards :
    - Espace
    - Virgule
    - Retour à la ligne ⇔ on peut lire d'un coup plusieurs lignes
  - Exemple :

    ➔ integer :: n =1, p = 2, q = 3

    ➔ read * n,p,q

Principe de lecture

- Primitive read :
  - Exemple :

    ➔ *integer :: n =1, p = 2, q = 3*

    ➔ *read * n,p,q*

    125,126,127 ⇔ n = 125, p = 126, q = 127

    125, , 127 ⇔ n = 125, p = 2, q = 127

    , , 127 ⇔ n = 1, p = 2 , q = 127

Lecture information condensée

- Valeur lue :

  N * value ⇔ $\underbrace{\text{value, value, …, value}}_{\text{N fois}}$

- Permet :
  - Le remplissage rapide de tableau
  - La diminution de la taille des informations sur les flux

Formatage de lecture/écriture

- Un format (en Fortran ⇔ >= 90)

  ⇔ chaîne de caractère de description du format attendu

  – *En FORTRAN (⇔ < 90) : on pouvait créer une entrée format avec étiquette à part*

- Syntaxe des formats numériques :
  – iw ⇔ i3 : entier sur 3 caractères
  – fw.d ⇔ f8.2 : réel sur 8 caractères dont 2 décimales
  – ew.d ⇔e12.4 : notation exponentiel sur 12 caractère dont 4 pour la mantisse

Formatage de lecture/écriture

- Syntaxe des formats numériques :
  - iw ⇔ i3 : entier sur 3 caractères
  - fw.d ⇔ f8.2 : réel sur 8 caractères dont 2 décimales
  - ew.d ⇔e12.4 : notation exponentielle sur 12 caractères dont 4 pour la mantisse
- Cas des espaces :
  - En Fortran (>=90) : ils sont éludés à la lecture dans le format
  - En FORTRAN (<90) : ils sont remplacés par des 0

Formatage de lecture/écriture

- Autres formats :
  - lw ⇔ l5 : booléen sur 5 caractères
  - wx ⇔ 3x : 3 espaces (éludés ou écrits)
  - tp ⇔ t10 : force le descripteur de ligne à se placer à une position particulière de la ligne ⇔ on se place au dixième caractère de la ligne en tampon
    - Possible car l'écriture sur les flux se fait ligne par ligne (et non au fur et à mesure)

Cas sur « / »

- Forcer le retour à la ligne à l'écriture

⇔ rajout d'un « \n » dans beaucoup de langages

- En Fortran, on met « / » (sans séparation)

- En lecture, cela permet de passer à la ligne suivante sans se préoccuper de la suite

Informations condensées (suite)

- On peut aussi mettre un multiplicateur sur les formats :
  - 4i5 ⇔ i5, i5, i5, i5
  - 3(i3, 5x, i2) ⇔ i3, 5x, i2, i3, 5x, i2, i3, 5x, i2
  - 3(i3, 2(i2, 3x, i4), i6)
    ⇔ i3, i2, 3x, i4, i2, 3x, i4 , i6, i3, i2, 3x, i4, i2, 3x, i4 , i6, i3, i2, 3x, i4, i2, 3x, i4 , i6

Cas des données > format

- A l'écriture, si le nombre de données est supérieur à la description du format
  - ⇔ Cas des tableaux
    - ➔ relecture du format pour finaliser

- Permet de formater simplement une sortie quelque soit la taille des données

Read & write

- L'instruction *write* est duale au *read* :
  - Écriture :
    - sur la sortie standard
    - formatée ou non : le non formatage correspond à *
    - Répétée tant que les données ne sont pas toutes consommées

Gestion des fichiers

- Ressemble au C ⇔ création d'un descripteur de fichier : c'est un nombre.

- Instruction *open* avec (ou non) le :
  - descripteur ⇔ paramètre *unit*
  - nom du fichier à ouvrir ⇔ paramètre *file*
  - format ⇔ paramètre *form*
  - status ⇔ paramètre *status*
  - retour d'ouverture ⇔ paramètre *iostat*

Gestion des fichiers

- Parcours classique de lecture :
  - Boucle infinie ⇔ *do … end do*
  - *read* avec fixation du paramètre *iostat*
  - Si iostat != 0 ➔ exit (sortie de la boucle uniquement)
  - fermeture de l'unité ⇔ *close*

LES TABLEAUX

Allocations statique et dynamique

Fonctionnement global

- A la manière des *ndarray* de Python (Numpy), il est possible, en Fortran, :
  - de gérer directement des « tranches » de tableaux
  - d'appliquer des traitements globaux sur des tableaux en une seule instruction :
    - Affectation de valeurs
    - Tests
  - de construire par bloc de tableaux multi-dimensionnels