

TP LC209 : Introduction à UNIX et FORTRAN, version 2005

P.Reinhardt

Laboratoire de Chimie Théorique, Université Pierre et Marie Curie – Paris VI,
4 place Jussieu, F – 75252 Paris CEDEX 05, France

Peter.Reinhardt@upmc.fr

Cette brève introduction à l'utilisation des machines IBM sous leur système d'exploitation AIX et quelques petits détails de la programmation en FORTRAN 77 a pour but d'aider aux travaux pratiques du module LC 209. Il ne remplacera pas un cours ni prétend d'être distribué à titre officiel.

Avis important: il n'y a jamais de solution unique dans la vie.

September 24, 2005

Table de Matières

I	Les commandes UNIX à utiliser	3
A	Au secours!!	4
B	Les commandes de gestion de fichiers	4
C	Les droits d'accès	5
D	Les commandes d'édition des fichiers	5
E	Récapitulatif des commandes UNIX utilisées	6
II	Le FORTRAN	7
A	Les débuts	7
1	Les types de variables, les opérations de base et la déclaration	8
2	Les comparaisons et les branchements	9
3	Tableaux et boucles	10
4	READ, WRITE et le FORMAT pour les entrées et les sorties	11
B	La suite	12
1	La représentation des entiers	12
2	Création et utilisation de fichiers	13
3	Appel de sous-programmes et de fonctions	14
4	COMMON blocks	15
5	Utilisation de bibliothèques de sous-routines	16
6	Les nombres complexes	16
7	Initialisation explicite de données	16
8	Une utilisation de GO TO	17
C	Récapitulatif des commandes FORTRAN	18
III	Exercices	19
1	Création et manipulation de fichiers	19
2	Calcul de π , première version	19
3	Calcul des fonction trigonométriques	19
4	Les nombres premiers, I	19
5	Les nombres premiers, II	19
6	Nombres complexes	20
7	Calcul de π par la formule de Stirling	20
8	Un petit programme autour du modèle de Bohr	21

PRÉLIMINAIRES

La chimie utilise beaucoup des logiciels comme boîtes noires pour calculer des propriétés des atomes et des molécules. Cependant, ces logiciels ont été écrits par des confrères chimistes, des algorithmes ont été développés par le besoin de diagonaliser par exemple des grandes matrices. Il est donc indispensable de connaître au moins quelques éléments de la programmation, soit juste pour redimensionner un logiciel de calcul à la taille d'un système à étudier ou pour vérifier l'implémentation détaillée d'une formule à évaluer. Un ordinateur ne peut fournir que ce qu'on demande, et toujours faut-il être vigilant avant de faire confiance aux résultats obtenus.

L'environnement UNIX a été développé¹ dans les années 1970 pour faire face à des possibilités de programmation interactive. Avant, c'était l'écriture d'un logiciel sur des cartes perforées, qui passaient par un lecteur et un ordinateur de centre de calcul pour être exécutées une seule fois. Alors, l'ALGOL, le COBOL et le FORTRAN (Formula Translator) étaient à l'ordre du jour, le dernier dans sa version FORTRAN IV et ensuite FORTRAN 77, qui est devenu quasiment standard dans les applications scientifiques. Aujourd'hui, le FORTRAN est remplacé par le C, C++, le FORTRAN 95 et d'autres langues plus élaborés.

Néanmoins, la simplicité des instructions dans le FORTRAN 77 est bien pratique pour écrire des petits projets, et surtout, la rapidité du code généré par les compilateurs (notamment des compilateurs d'IBM), le rend encore préférable au C, quand il s'agit de grandes demandes de calcul. Notons que le FORTRAN est utilisé pour calculer des nombres ; faire du graphisme ou traitement de textes, la programmation de cela on laisse aujourd'hui aux langues plus adaptées à l'interaction avec l'utilisateur.

I. LES COMMANDES UNIX À UTILISER

L'UNIX sert à diriger l'ordinateur vers ce qu'on voudrait lui faire faire. On parle donc d'un système d'exploitation qui a, dans le cas d'UNIX, l'avantage d'être présent sur des ordinateurs de toute taille, du portable aux super-calculateurs des plus grands centres de calcul nationaux. Des tâches standards sont la création des répertoires (en anglais : *directories*), la gestion des fichiers, la création des fichiers de données pour des programmes scientifiques, l'impression des résultats, l'archivage et le transfert des données et le traitement graphique des résultats.

Les distributions LINUX disposent d'une surface graphique en général, avec des icônes connues des PC sous Windows ou MacOS. Mais il faut savoir que derrière toute commande iconisée il y a une commande UNIX qu'on peut lancer directement à partir d'une fenêtre de commandes ou d'un écran qui ne dispose pas d'une surface graphique.

Les commandes UNIX sont rarement évidentes, même si on est expert de la langue anglaise. Nous n'en rencontrerons que quelques commandes dans ce parcours initiatique. Une fois perdu, ou ayant oublié la syntaxe exacte d'une commande, la commande

man (**commande**) peut aider à retrouver les détails. Il existe d'ailleurs une centaine de "dialects" UNIX différents, chaque constructeur d'ordinateur a un peu développé le sien. Ce que nous utilisons est l'UNIX de IBM, appelé AIX, et le compilateur Fortran d'IBM, le **xlf**.

A. Au secours!!

L'IBM du CICRP ne connaît pas d'action liée à la touche ←. Pour qu'elle efface quelque chose il faut soit utiliser **<CTRL> h**, soit l'activer par **stty erase ←**.

S'il y a du bazar sur l'écran, écrit par un autre programme par exemple, on revient à un écran propre avec **<CTRL> l**.

Un programme en exécution qui a dérapé, est arrêté instantanément avec **<CTRL> c**. **<CTRL> s** bloque le clavier; si cela vous arrive, vous débloquent le clavier par **<CTRL> q**.

B. Les commandes de gestion de fichiers

Sur un disque dur, les fichiers sont organisés dans des répertoires avec une arborescence.

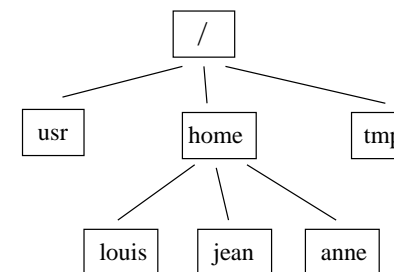


FIG. 1. Un exemple d'une arborescence UNIX

On change de répertoire avec la commande **cd**. Soit par rapport au répertoire actuel (son nom se dévoile avec **pwd** comme "present work directory") par **cd ..** ou **cd mon-sous-répertoire**, soit avec son identification absolue par **cd /home/mon_login/mon-répertoire/mon-sous-répertoire**. Le contenu d'un répertoire est affiché par la commande **ls** avec des options : par exemple **ls -l** affiche les détails, **ls -lt** affiche les fichiers dans un ordre chronologique, **ls -F** ajoute des identifiants aux types de fichiers etc. **mkdir mon-sous-répertoire** crée un nouveau répertoire, **rm mon-fichier** efface "mon-fichier" et **rm -r mon-sous-répertoire** efface tout le sous-répertoire.

Évitez les blancs dans les noms des fichiers ou répertoires. De même les accents sont très mal vus en UNIX !

Pour copier un fichier nous utilisons la commande **cp**. Copier des répertoires entiers se fait avec l'option **cp -R**. On déplace ou renomme un fichier ou répertoire par **mv**.

¹Plus d'information sur l'histoire : <http://www.levenez.com/unix/> et <http://www.princeton.edu/~mike/expotape.htm>

C. Les droits d'accès

Avec la commande **ls -l** un affichage en plusieurs colonnes est produite : d'abord 10 caractères, ensuite un nombre, ensuite le nom du propriétaire et le groupe, dont il appartient, la taille des fichiers, la date de leur dernière modification et, finalement, leur nom. Les premiers 10 caractères donnent le type du fichier et les droits à son accès. La première lettre est un “-” pour un fichier, un “d” pour un répertoire et un “l” pour un lien symbolique. Après il y a trois groupes de 3 lettres “rwx” qui signifient le droit de lecture, d'écriture et d'exécution pour l'utilisateur, le groupe, dont il appartient, et le reste du monde. Ces droits peuvent être manipulés par la commande **chmod** et un code en système binaire : pas de droit = 0, droit = 1, alors par exemple r-x = 101 (oct.) = 5. Nous voudrions donner accès à tout le monde pour lecture et exécution, mais garder le droit d'écriture exclusivement pour nous ? Alors c'est **chmod 755**. Le contenu d'un répertoire n'est visible qu'à condition qu'on ait le droit de son exécution. Donc mettre **chmod 644** ne suffit pas pour rendre le contenu accessible, il faudrait plutôt **chmod 755**.

D. Les commandes d'édition des fichiers

Les experts sous UNIX utilisent **emacs** ou **vi**. Hélas, **emacs** n'est pas disponible partout, et **vi** ressemble sans expérience à un tour de magie. Nous utilisons alors un éditeur graphique, **nedit**. Dans une fenêtre apparaît une page vide, avec une barre de menus en haut.

Dans **Preferences** on choisit **Language modes**, puis **Fortran**, et tout est prêt pour programmer en **FORTRAN**. Les autres menus sont relativement explicites dans leur contenu. On est prêt pour se lancer dans la programmation.

E. Récapitulatif des commandes UNIX utilisées

- **cd** : changer de répertoire
- **ls** : afficher le contenu d'un répertoire, avec les options
 - l : afficher les détails
 - t : par ordre chronologique
 - R : en parcourant l'arborescence
 - a : en affichant des fichiers cachés, dont le nom commence par un point.
 - F : afficher des informations sur le type de fichier ou répertoire.
- **mkdir** : créer un répertoire
- **mv** : déplacer ou renommer un fichier ou répertoire
- **cp** : copier un fichier ou répertoire
- **rm** : effacer un fichier ; **rm -r** : effacer un répertoire
- **du** : afficher l'utilisation de disque par le répertoire actuel ; options
 - s : afficher que le total
 - k : afficher les résultats en kilo-octets (en blocs de 512 octets par défaut sinon)
- **df** : afficher l'utilisation de disque. C'est bien pour vérifier, si par hasard ou inadvertance un disque est saturé.
- **ps x** : affichage de ses processus actifs.
- **top** : afficher les processus les plus actifs du moment.
- **kill -9** : tuer un processus actif.
- **vi** : le programme d'édition de fichiers
- **cat** : afficher le contenu d'un fichier
- **tee** : faire sortir l'affichage sur écran également dans un fichier ; à utiliser avec le | (anglais : *pipe*), par exemple **ls -l | tee contenu_de_mon_répertoire**.
- * remplace une chaîne de caractères, ? remplace un seul caractère
- **f77** ou **xlf** : invocation du compilateur **FORTRAN**
- < redirection d'une entrée (on lit les données d'un fichier par **programme < données**)
> redirection de la sortie vers un fichier (e.g. **programme < données > sortie**).

II. LE FORTRAN

A. Les débuts

Un programme FORTRAN est une liste ou collection d'instructions, à exécuter dans l'ordre d'apparence. Une règle générale : une ligne d'un programme ne contient qu'une seule instruction et commence toujours par au moins 6 blancs. Il y a trois exceptions : soit il s'agit d'une ligne de continuation, parce que la ligne avant était trop longue (le FORTRAN standard ne prend que les 72 premiers caractères d'une ligne). Dans ce cas il y a un caractère (n'importe lequel) sur la sixième position. Soit il s'agit d'une ligne numérotée (en anglais : *label*). Le numéro se trouve alors sur les premiers cinq positions. Ou bien c'est un commentaire, pour lequel il se trouve un C en première position. Exemples:

```
      WRITE(6,*) ' we write here a general result in detail',  
-      vect(i),mat(m),result(vect(j(2*i+m)))  
C  
C here we contract 4 elements of an array  
C  
      DO 10 I=1,200,4  
        MAT(I)=MAT(I)+MAT(I+1)+MAT(I+2)+MAT(I+3)  
10 CONTINUE
```

Un programm FORTRAN commence par une première ligne

```
PROGRAM MAIN
```

et se termine par

```
END
```

Donc un programme FORTRAN minimal est déjà

```
PROGRAM MAIN  
WRITE(6,*) ' On commence le FORTRAN'  
END
```

Une fois ce programme écrit, on le transforme en exécutable par un compilateur. Pour que le compilateur sache qu'il s'agit d'un programme FORTRAN, il est utile de lui donner un nom avec une extension `.f`, par exemple `simple.f`. La commande `xl f simple.f` le transforme alors en `a.out`. Avec l'option `-o` on peut choisir un autre nom pour l'exécutable, sinon au bout de quelques essais on se retrouve qu'avec des `a.out` dont on ne connaît plus leur fonctionnement. Par exemple `xl f -o simple simple.f`.

Le programme compilé `simple` est maintenant prêt pour être lancée, comme n'importe quelle commande UNIX.

1. Les types de variables, les opérations de base et la déclaration

FORTRAN connaît les nombres entiers, les nombres réels, les nombres complexes, les variables logiques et des caractères. Pour attribuer une valeur à une variable, on écrit `variable=valeur`, par exemple

```
A=1.0  
I=3  
ZF=5.3D3  
L17=.FALSE.  
ASTR='BEURK'
```

Le type du résultat dépend du type de la variable utilisée : si la valeur donnée était un nombre entier (*integer*) et la variable est de type "double précision" (*double precision*), FORTRAN fait automatiquement une promotion `integer` → `double precision`. Mais attention, l'inverse est vrai aussi : si la valeur était `15.56718D+00` (`double precision`) et la variable du type `integer`, il y aura que 15 retenu, les chiffres derrière la virgule seront négligés (pas arrondis!).

Comme FORTRAN signifie Formula Translator, il y a toutes les opération mathématiques disponibles : `+`, `-`, `*`, `/`. Mais aussi les fonctions standards comme `exp`, `sin`, `abs` etc. sont disponibles et font partie du langage. Pour des tâches plus compliquées (par exemple l'inversion d'une matrice) on fait appel à des routines déjà écrites ou des sous-programmes.

Ensuite, il y a les expressions logiques du type (variable logique) = (un test ou la valeur d'une autre variable logique). Les tests logiques sont `.EQ.`, `.NE.`, `.GE.`, `.GT.`, `.LE.` et `.LT.` (*equal* (=), *not equal* (≠), *greater or equal* (≥), *greater than* (>), *less or equal* (≤) et *less than* (<)) entre nombres et `.EQV.` et `.NEQV.` entre variables ou valeurs logiques. En plus nous disposons de la négation `.NOT.` et des opérateurs logiques `.AND.` et `.OR.`. Les points font partie de la syntaxe, nous pouvons alors avoir des expressions comme `LI=(J.EQ.1).AND..NOT.(K.EQ.2)` avec une variable logique LI et deux variables `integer` J et K.

Le type d'une variable est à déclarer au début d'un programme par des lignes

```
PROGRAM MAIN  
INTEGER I,K  
LOGICAL LI  
DOUBLE PRECISION AVAL,RESULT  
REAL REALVAL  
CHARACTER*7 FILNAM  
...  
...  
END
```

qui concernent le morceau entre `PROGRAM` et `END`. Le FORTRAN d'IBM attribue par défaut le type `INTEGER` aux variables commençant par I..N et le type `REAL` aux variable commençant avec les autres lettres de l'alphabet (d'ailleurs, les IBM acceptent le \$ comme une lettre de l'alphabet). Il est alors **vivement recommandé** de clarifier la situation au début

pour qu'un programme soit transportable d'une machine à une autre sans provoquer des confusions. Soit on déclare toutes les variables explicitement en forçant la non-attribution d'un type par une ligne

```
IMPLICIT NONE
```

avant la liste des variables, soit on choisit par exemple

```
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
IMPLICIT INTEGER (I-N)
```

ou

```
IMPLICIT REAL*8 (A-H,O-Z)
```

pour détailler la précision souhaitée des variables. Le 8 signifie ici que le compilateur doit réserver 8 octets (64 bit) pour chaque variable (e.g. un bit pour le signe, 54 bit pour les chiffres, 8 bits pour l'exposant et 1 bit pour son signe).

$$146.678047E-05 = 1.46678047 \times 10^{-3} \rightarrow \underbrace{+}_{\text{signe}} \underbrace{146678047}_{\text{chiffres}} \underbrace{-}_{\text{signe}} \underbrace{3}_{\text{exposant}}$$

Sur une machine IBM REAL réserve 4 octets et DOUBLE PRECISION 8 octets, sur une CRAY par contre déjà le REAL occupe 8 octets.

Toute variable déclarée séparément impose son type au dessus d'une déclaration générale par IMPLICIT.

2. Les comparaisons et les branchements

Les comparaisons pour déclencher un branchement sont de la forme

```
IF (valeur logique) THEN
...
ELSE
...
END IF
```

avec les opérateurs logiques que nous avons vus. S'il n'y a qu'une commande à exécuter on peut utiliser la forme alternative

```
IF (A.GT.1.DO) B=SIN(X)
```

sans THEN et END IF (ou ENDIF, les blancs sont ignorés). La valeur logique peut également être la valeur d'une variable logique, par exemple IF (LI) THEN ... ou bien IF (LI.AND..NOT.LJ) THEN

On trouve encore souvent des branchements avec une commande GO TO et une ligne CONTINUE de la forme

```
IF (I.EQ.1) GO TO 100
   lignes de commandes
   GO TO 200
100 CONTINUE
   autres lignes de commandes
200 CONTINUE
```

qui ne sont rien d'autre que

```
IF (I.EQ.1) THEN
   autres lignes de commandes
ELSE
   lignes de commandes
END IF
```

plus élégant et moins difficile à lire. Ce n'est pas le GO TO qui pose le problème, ce sont plutôt les lignes 100 CONTINUE et 200 CONTINUE qui introduisent la confusion, parce qu'on ne sait pas où le programme est passé avant d'atterir au label 200. Avait-il peut-être une commande GO TO 200 ailleurs? Evitons les lignes CONTINUE.

S'il y a plusieurs possibilités, on peut soit imbriquer les IF ... END IF, soit utiliser la commande ELSE IF (condition) THEN, par exemple

IF (I.EQ.1) THEN		IF (I.EQ.1) THEN
commande(s) 1		commande(s) 1
ELSE IF (I.EQ.2) THEN		ELSE
commande(s) 2		IF (I.EQ.2) THEN
ELSE IF (I.EQ.3) THEN		commandes(s) 2
commande(s) 3		ELSE
ELSE		IF (I.EQ.3) THEN
WRITE(6,*) ' choix impossible pour I'		commandes(s) 3
STOP		ELSE
END IF		...

3. Tableaux et boucles

Une matrice peut être déclaré comme un tableau, par exemple

```
DOUBLE PRECISION AMAT(100,10,23)
```

ou

```
DOUBLE PRECISION AMAT
PARAMETER (NDIMX)
DIMENSION AMAT(NDIMX,NDIMX)
```

La convention de FORTRAN sur le stockage d'une matrice en mémoire et sur disque est la suivante :

```
AMAT(1,1) ... AMAT(100,1) AMAT(1,2) ... AMAT(100,2) ... AMAT(100,100)
```

Si la matrice est un ensemble de vecteurs de longueur 3 par exemple, on peut éviter que l'ordinateur est obligé de faire des sauts en mémoire en déclarant la matrice comme `AMAT(3,NVECT)`. Une matrice allant de -10 à $+10$ (21 éléments) est définie par `DIMENSION AMAT(-10:10)`.

Quelques possibilités d'utilisation des éléments de matrices:

```
AMAT(1,5)=10.DO
VALMAT=AMAT(2,3)
AMAT(1,1)=AMAT(2,3)+AMAT(2,4)
AMAT(1,1)=AMAT(1,1)+AMAT(1,2)+AMAT(1,3)
```

Pour adresser les éléments d'une matrice ou pour écrire des boucles plus générales, il y a la structure `DO variable=début, fin, incrément ... END DO`, par exemple

```
DO K=1,100,4
...
les commandes
...
END DO
```

qui veut dire que les commandes à l'intérieur de la boucle seront exécutées pour $K=1$, ensuite 5, 9, 13, etc. jusqu'à $K=97$, puisque le prochain K , 101, est au delà de 100. L'incrément (ici 4) est par défaut 1 et il n'est pas obligatoire. D'ailleurs, il peut être négatif comme `DO K=100,1,-4`. Les structures de boucles `REPEAT ... UNTIL` et `DO WHILE ... END DO`, connues peut-être du PASCAL ou C, ne sont pas définies sur tous les dialects de FORTRAN 77.

On peut utiliser également des *labels* pour terminer une ou plusieurs boucles :

```
C nous parcourons le triangle K>=L d'une matrice symetrique
C la matrice est dans un tableau uni-dimensionnel
I=0
DO 100 K=1,100
DO 200 L=1,K
I=I+1
WRITE(6,*) ' Matrix element No ',K,L,' = ',AMAT(I)
200 CONTINUE
100 CONTINUE
```

Moins à recommander est de terminer plusieurs boucles par un seul *label*, en écrivant `DO 100 L=1,K` dans l'exemple.

4. READ, WRITE et le FORMAT pour les entrées et les sorties

La lecture des données et l'écriture des résultats se fait par des commandes `READ` et `WRITE` et des **unités logiques**. L'unité logique de lecture par défaut est le canal 5, et d'écriture le canal 6. Si un programme FORTRAN se lance comme **commande** `< input > output`, il y a à l'intérieur des lignes de lecture de forme

```
READ(5,*) DATA1,DATA2,DATA3
```

et d'écriture

```
WRITE(6,*) RESULT1,RESULT2,RESULT3
```

L'astérisque `*` signale que la lecture et l'écriture se font dans un format libre, c'est-à-dire qu'on n'impose pas le format, sous lequel doivent se trouver `DATA1`, `DATA2`, `DATA3`. Le FORTRAN lit jusqu'à ce que il y ait trois données ou une erreur de lecture (fin de fichier, pas de nombre mais caractères) en utilisant le fichier `input`. La sortie peut être mise en forme par des lignes définissant un format, par exemple :

```
IOUT=6
WRITE(IOUT,50) I, (A(J), J=INZ1, IFN1)
50 FORMAT(/,I4,3X,10(E12.4))
```

Nous nous attendons alors à une ligne blanche (`/`), un nombre entier sur 4 places (`I4`), trois blancs (`3X`) et jusqu'à 10 nombres réels sur 12 positions, avec 4 chiffres derrière la virgule et en donnant l'exposant comme "`_ 1.2345E+02`". Si l'on voudrait "`_ 123.45`", il fallait spécifier `F8.2` au lieu de `E12.4`. Remarquons qu'il y a un label associé à la ligne définissant le format.

Nous pouvons également inclure le format dans la ligne `WRITE` par

```
WRITE(6, '(I4,10(E12.4))') I, (A(J), J=INZ1, IFN1)
```

Des mots et phrases peuvent être inclus via

C attention aux apostrophes !!

```
WRITE(6,*) ' Ceci est un test de l''écriture d''une phrase '
```

B. La suite

Nous pouvons commencer à écrire des programmes à partir de ces éléments de base du FORTRAN. Mais rapidement il nous manquerait quelque chose pour ne pas écrire plusieurs fois les mêmes instructions dans des endroits différents des programmes. Ce chapitre donnera des possibilités supplémentaires. Avant d'étendre les connaissances de la langue, une petite excursion vers la précision numérique.

1. La représentation des entiers

Nous illustrons la réservation de la place en mémoire pour des entiers par le calcul d'un factoriel.

Prenons le programme suivant:

```
PROGRAM MAIN
IMPLICIT NONE

INTEGER*2 I2
```

```

INTEGER*4 I4,I
INTEGER*8 I8

INTEGER*8 IFACT

IFACT=1
DO I=1,20
  IFACT=IFACT*I
  I2=IFACT
  I4=IFACT
  I8=IFACT
  WRITE(6,9001) IFACT,I2,I4,I8
9001  FORMAT(4I20)
END DO
END

```

Nous voyons que l'exécution nous ne donne un résultat correct en utilisant `INTEGER*8`, à savoir une représentation d'un entier sur huit octets.

2. Création et utilisation de fichiers

Nous avons vu comment la lecture via le canal 5 et l'écriture via le canal 6 sont organisées. Cependant, il peut nous arriver d'avoir calculé les valeurs d'une quantité, que nous ne voulons pas recalculer, mais lire directement du fichier des résultats du calcul précédent. Ceci est possible avec l'ouverture des canaux (ou unités logiques) supplémentaires. Un exemple montrera l'utilisation :

```

OPEN(UNIT=17,FILE='RESULT.DAT',FORM='FORMATTED',STATUS='OLD')
C lecture via 2 boucles imbriquées
READ(17,*) ((AMAT(I,J),J=1,10),I=1,10)
CLOSE(17)
C le carré de la matrice AMAT
DO I=1,10
  DO J=1,10
    CMAT(I,J)=0.DO
  END DO
  DO K=1,10
    AIK=AMAT(I,K)
    DO J=1,10
      CDUM=0.DO
      CMAT(I,J)=CMAT(I,J)+AIK*AMAT(K,J)
    END DO
  END DO
END DO
C écriture du produit de la matrice AMAT^2 via 2 boucles imbriquées
OPEN(UNIT=17,FILE='CARRE.DAT',FORM='FORMATTED',STATUS='UNKNOWN')

```

```

WRITE(17,'(4E20.11)') ((AMAT(I,J),J=1,10),I=1,10)
CLOSE(17)

```

Il est aisé de fermer le fichier directement après lecture ou écriture, sinon le canal 17 sera bloqué pendant toute l'exécution du programme. Du temps des bandes magnétiques comme moyens de stockage des données vient l'instruction `REWIND` pour rebobiner un canal de lecture, pour re-lire l'information une deuxième fois sans fermer et rouvrir le fichier :

```

OPEN(UNIT=17,FILE='RESULT.DAT',FORM='FORMATTED',STATUS='OLD')
C lecture dans matrice AMAT
READ(17,*) ((AMAT(I,J),J=1,10),I=1,10)
REWIND(17)
C lecture dans matrice BMAT
READ(17,*) ((BMAT(I,J),J=1,10),I=1,10)
CLOSE(17)

```

Si on ne donne pas de nom explicite, FORTRAN va chercher un fichier `fort.17` associé au canal 17. En absence de ce fichier, le programme s'arrête avec un message d'erreur, parce qu'on a spécifié `OLD`, c'est-à-dire que le fichier doit exister pour le bon fonctionnement.

3. Appel de sous-programmes et de fonctions

Une fois un petit morceau de programme pour une tâche précise écrit, nous pouvons le mettre à part dans une `SUBROUTINE` ou `FUNCTION`. Les données passent par des listes d'appel. Par exemple, la lecture de la matrice via canal 17 peut se faire dans une sous-routine :

```

...
CALL RDMATQ(AMAT,10)
...

SUBROUTINE RDMATQ(X,NDIM)
  IMPLICIT NONE
  INTEGER NDIM,I,J
  DOUBLE PRECISION X(NDIM,NDIM)
C
  OPEN(UNIT=17,FILE='RESULT.DAT',FORM='FORMATTED',
    -      ,STATUS='OLD',ERR=800)
C lecture dans matrice X
  READ(17,*) ((X(I,J),J=1,NDIM),I=1,NDIM)
  CLOSE(17)
  RETURN
C sortie d'erreur
800  CONTINUE
  WRITE(6,*) ' pas de fichier <RESULT.DAT> '
  STOP
END

```

L'appel se fait par CALL et ce qui est donné à la routine sont les adresses des arguments en mémoire, pas leur type ni leurs dimensions. Tout doit être correctement déclaré dans l'entête de la routine et nous pouvons communiquer la bonne dimension d'une matrice par un argument dans la liste d'appel. Certains compilateurs se plaignent s'il y a un argument dans le CALL d'un type et dans la déclaration d'un autre type, mais ce n'est pas garanti.

Une fonction, par contre, est appelée directement et fournit une valeur :

```
...
X=RACINE4(Y)
...

FUNCTION RACINE4(X)
IMPLICIT NONE
DOUBLE PRECISION X,RACINE4
RACINE4=SQRT(SQRT(X))
RETURN
END
```

Donc un programme FORTRAN consiste en général d'un programme maître et des sous-routines et fonctions. Mentionnons qu'en FORTRAN des recursions (appel d'une fonction à l'intérieur d'elle-même) sont interdites. Toutes les variables définies à l'intérieur d'une fonction ou d'une sous-routine sont locales et en général perdues dans un deuxième appel de la routine. Pour garder certaines informations il y a une structure spéciale, les COMMON blocks.

4. COMMON blocks

Souvent il arrive le cas qu'il y a plusieurs quantités à précalculer et à stocker comme des constantes. On peut réserver de la mémoire pour ces quantités en définissant un COMMON BLOCK :

```
FUNCTION CIRC(R)
IMPLICIT NONE
COMMON /CONSTS/ PI,TWOPI
DOUBLE PRECISION PI,TWOPI,R,CIRC
CIRC=2.DO*PI*R
RETURN
END
```

qui est mis avant la déclaration des variables. Evidemment, ces valeurs doivent être attribuées au début du programme maître par

```
PROGRAM MAIN
IMPLICIT NONE
COMMON /CONSTS/ PI,TWOPI
DOUBLE PRECISION PI,TWOPI
```

```
PI=2.DO*ACOS(0.DO)
TWOPI=2.DO*PI
```

5. Utilisation de bibliothèques de sous-routines

Un sous-programme ou une fonction peuvent être compilés et gardés dans une version compilée à part, en spécifiant l'option -c lors de l'appel du compilateur. En mettant la fonction RACINE4 dans un fichier séparé `racine4.f` et en lançant `xlf -c racine4.f` nous créons un fichier `racine4.o`. Si nous avons besoin de la routine, nous la ajoutons à un exécutable par `xlf program.f racine4.o`. Il existe des bibliothèques de routines (un serveur étant <http://www.netlib.no>) avec des sources à télécharger et à compiler soi-même. Si on connaît l'ordre correct de l'appel d'une sous-routine d'une bibliothèque, on peut utiliser également des bibliothèques précompilées et **optimisées** pour sa machine, surtout pour des tâches chères en calcul, mais avec une finalité limitée comme les transformations de Fourier ou les multiplications et la gestion des matrices.

6. Les nombres complexes

FORTRAN connaît le type COMPLEX et les variables déclarées de ce type sont stockées comme deux variables réelles. La partie réelle de la variable A est extraite par REAL(A) et la partie imaginaire par IMAG(A). L'attribution d'une valeur se fait par exemple en donnant une paire de nombres réels par `A=(1.DO,3.DO)`. Il faut toujours savoir qu'une variable complexe demande deux fois plus de place en mémoire qu'une variable réelle. Nous voyons aussi pourquoi l'attribution du type REAL ne se fait pas par REAL(N), mais par FLOAT(N).

La multiplication et l'addition des nombres complexes se fait souvent plus rapide en programmant les opérations explicites avec des nombres réels,

$$(a + ib)(c + id) = (ac - bd) + i(ad + bc)$$

parce que le type COMPLEX n'était jamais trop la préoccupation des constructeurs des compilateurs.

7. Initialisation explicite de données

L'initialisation des constantes peut se faire par une ligne DATA :

```
PROGRAM MAIN
COMMON /CONSTS/ HALF,ONE,TWO,THIRD
COMMON /ORBITS/ CORB(0:5)
DOUBLE PRECISION HALF,ONE,TWO,THIRD
CHARACTER*1 CORB
DATA HALF /0.5DO/,ONE /1.DO/, TWO /2.DO/, THIRD /0.3333333DO/
DATA CORB /'S','P','D','F','G','H'/
```


Cette initialisation, si la liste est longue, peut être mise dans une routine qui ne fait rien d'autre et qui prend la forme d'un `BLOCK DATA`. C'est comme un programme à part, avec en première ligne `BLOCK DATA` et en dernière `END`, sans véritable instruction à l'intérieur :

```

BLOCK DATA
IMPLICIT REAL*8 (a-h,o-z)
COMMON /adres/iadr(50)
COMMON /locij/nlocij,locij(20000)
COMMON /indx/indx(17)/label/title(60),xtit(30),
$ /fenerg/eint(50)/io/iunit(4),nrec(4),nio/vcrit/vcrit
COMMON /etime1/time(250),r1(30),r2(30),r3(30),r4(30),r5(30),
$ r6(50),r7(50),ienter(250)
CHARACTER*8 title,xtit,r1,r2,r3,r4,r5,r6,r7
DATA iadr/50*0/,indx/17*0/,ipath/1000*0/,time/250*0.d00/,
$ ienter/250*0/,iunit/1,2,3,12/,nrec/4*0/,ie14p1/8*0/
DATA locs/80*0/,locd/80*0/,loct/510*0/
C --- Integral labels ---
DATA title/'AAAA0000','AAAA000V','AAAA00VV','AAAA0VOV','AAAA0VVV',
$ 'AAAAVVVV',
...
END

```

Cette structure est utile par exemple pour des intégrations numériques ou des formules d'interpolation dans un polynôme donné.

8. Une utilisation de `GO TO`

Lorsque l'ordinateur lit un fichier, il le lit ligne par ligne et essaye d'attribuer les données lues aux variables destinées. Mais comment le programme sait qu'il faut s'arrêter de lire ? Il faut un condition, qui est implémenté par une variable `IOSTAT`. Son utilisation est la suivante, un peu exceptionnelle :

```

100 CONTINUE
   READ(17,*,IOSTAT=K) XVAL
   IF (K.NE.0) GO TO 200
   ...
   instructions
   ...
   GO TO 100
200 CONTINUE

```

La variable `K` prend la valeur de l'état d'avancement de la lecture, contraire à l'instruction habituelle `K=IOSTAT`. La variable `IOSTAT` est zéro pour lecture correcte, et prend une autre valeur (dépendant du FORTRAN du constructeur) si une erreur se produit, ici la fin du fichier lié au canal 17. Alors on saute vers la ligne numérotée 200.

C. Recapitulatif des commandes FORTRAN

- `PROGRAM`, `BLOCK DATA`, `SUBROUTINE`, `FUNCTION`, `END`, `STOP`, `RETURN` : structuration d'un programme
- `IF`, `THEN`, `ELSE`, `END IF` : branchement ou aiguillage
- `DO`, `END DO` : boucle
- `WRITE`, `READ` : lecture et écriture ; unité logique ou canal 5 est réservé d'office pour la lecture interactive et canal 6 est réservé pour l'écriture sur l'écran.
- opérateurs + * / -
- opérateurs logiques `.EQ.`, `.NE.`, `.GE.`, `.GT.`, `.LE.`, `.LT.`, `.NOT.`, `.AND.` et `.OR.`
- `GO TO`, `CONTINUE` : sauts dans le programme (à éviter)
- `PARAMETER` : dimensions générales
- `COMMON` : place en mémoire réservée en permanence
- `DATA`, `BLOCK DATA` : initialisation explicite de variables
- `IMPLICIT`, `INTEGER`, `REAL`, `DOUBLE PRECISION`, `LOGICAL`, `CHARACTER`, `COMPLEX` : types de données et variables
- `DIMENSION` : taille des tableaux
- `OPEN`, `CLOSE`, `REWIND` : manipulation de fichiers

III. EXERCICES

1. Création et manipulation de fichiers

Donner la séquence de commandes UNIX pour créer un répertoire, un fichier dans ce répertoire, renommer le fichier, changer le droit d'accès pour que tout le monde puisse le lire et exécuter mais que vous et votre groupe seulement puissiez le modifier.

2. Calcul de π , première version

La série

$$y(n) = \sum_{i=1}^n \frac{1}{i^2}$$

converge (lentement) vers $\pi^2/6$ avec $n \rightarrow \infty$. A combien faut-il pousser n pour obtenir π à 5 chiffres près ? Comparer avec π calculé par **PI=2.D0*ACOS(0.D0)**.

3. Calcul des fonction trigonométriques

Nous connaissons les séries pour représenter **sin(x)** et **cos(x)** :

$$\begin{aligned} \sin(x) &= \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!} \\ \cos(x) &= 1 + \sum_{n=1}^{\infty} (-1)^n \frac{x^{2n}}{(2n)!} \end{aligned} \quad (1)$$

Comparer le calcul par la fonction standard du FORTRAN et l'implémentation (en **double precision**) de ces séries par rapport à leur comportement

- La précision pour des valeurs de x différentes, entre zéro et 2π . Calculer π auparavant par **PI=2.D0*ACOS(0.D0)**. Pour avoir une précision au delà de 10^{-7} , combien d'éléments de la somme sont en moyenne à retenir? Suffira-t-il de couper la sommation après $n = 4$?

4. Les nombres premiers, I

Décomposer un nombre entier entre 2 et 1000 en ses nombres premiers.

5. Les nombres premiers, II

Ecrire un programme en plusieurs étapes :

- Nous connaissons les nombres premiers 2, 3, 5, 7, 11, 13, 17, 19, 23, 29 et 31.
- Faire calculer à partir des ces données (instruction **DATA**) les nombres premiers entre 2 et 997 et les mettre dans un fichier. Il y a plusieurs manières de le faire.

- Continuer ensuite le programme pour demander d'entrer un nombre et de l'analyser en lisant chaque fois les données stockées sur le fichier créé.

Le programme devrait avoir un structure

```
PROGRAM MAIN
...
DATA ??? /.../
...
C creation du fichier des donnees
OPEN(UNIT=...,FILE=..., ...)
WRITE(... )
CLOSE(UNIT=...)
C
C boucle sur des demandes interactives
C
OPEN(UNIT=... )
100 CONTINUE
WRITE(6,*) ' Donner un nombre entre 3 et 1000000'
READ(5,*) N
C nous cherchons si N est nombre premier
IF (N.EQ.0) THEN
CLOSE(UNIT=... )
STOP
END IF
...
REWIND(UNIT= ...)
GO TO 100
END
```

6. Nombres complexes

Trouver pour un nombre complexe $a + ib$ sa représentation en coordonnées polaires :

$$a + ib = r e^{i\phi}$$

Utiliser une arithmétique réelle (**REAL**) et une arithmétique complexe (**COMPLEX**).

7. Calcul de π par la formule de Stirling

Une bonne approximation du factoriel $N!$ est la formule de Stirling

$$\ln(N!) = N \ln N - N + \frac{1}{2} \ln(2\pi N)$$

qu'on peut utiliser pour calculer π par

$$y = \frac{N}{2} \left[(N-1)! \left(\frac{e}{N} \right)^N \right]^2 \xrightarrow{N \rightarrow \infty} \pi$$

Cette exercice montrera également les dangers numériques, puisque la formule est valable pour des grandes valeurs de N , mais le calcul de l'exponentiel limite N à une certaine taille. Essayer pour des valeurs de N entre 10 et 5000. Où s'arrête l'applicabilité de la formule en double précision?

8. Un petit programme autour du modèle de Bohr

Ecrire un programme (PROGRAM, SUBROUTINE, et FUNCTION) pour demander à l'utilisateur de spécifier un numéro atomique et deux niveaux n et m d'une raie lumineuse correspondant à la transition $n \rightarrow m$. Faire calculer sa longueur d'onde (en Ångström) en utilisant la constante de Rydberg (109700 cm^{-1}). Afficher les résultats avec deux chiffres derrière la virgule.

Modifier le programme pour calculer les longueurs d'onde des premiers 30 éléments d'une série $n \rightarrow n+1$ jusqu'à $n \rightarrow n+30$. Donner dans la sortie du programme également la raie limite ($m = \infty$ approché par $m = 150000$).

Prévoir que le programme ne s'arrête pas après une seule exécution, mais revient au début en demandant des données. Prévoir une sortie correcte pour arrêter l'exécution du programme.

La formule pour calculer la longueur d'onde λ :

$$\frac{1}{\lambda} = R \left(\frac{1}{n^2} - \frac{1}{m^2} \right)$$