UNIVERSITE PIERRE ET MARIE CURIE

Licence de Chimie UE-209

INTRODUCTION A LA PROGRAMMATION FORTRAN

Plan :

1 : Introduction

2 : Structure d'un programme Fortran

3 : Aiguillages

4 : Processus itératifs

5 : Variables indicées

6 : Sous-programmes

Contrôle des connaissances :

CC N°1 (*10 min - 5pts*) pendant le deuxième cours

CC N°2 (*10 min - 5pts*) pendant le troisième cours

Examen (*1h - 30pts*) pendant le quatrième cours

Objectifs

L'objectif de cette UE est d'approfondir les connaissances des étudiants dans les objets et concepts mathématiques de base utilisés dans les différents domaines de la chimie. L'enseignement est donc axé sur les besoins des différents enseignements dispensés en L2 et L3 de la mention chimie, dont seront tirées les principales applications. Les concepts sont illustrés et complétés par des enseignements d'informatique qui vont familiariser les étudiants avec des notions de Linux et de Fortran. Ces notions d'informatique permettront ensuite aux étudiants lors de leurs études (L3 et Master) d'aborder l'informatique appliquée aux sciences chimiques.

Conventions utilisées

Le texte écrit comme ceci est un message affiché sur l'écran ou un texte à taper au clavier

Le texte écrit *«comme ceci»* est un texte que vous devez adapter selon le contexte.

la touche ENTER (entrée) est indiquée par le signe ↵

le caractère espace est indiqué par le signe ^

- COURS 3 -

AIGUILLAGES

III.1) Ruptures de Séquence : les étiquettes.

Un programme fortran prend les instructions dans l'ordre successif des lignes. Il peut être intéressant de faire passer d'une ligne à une autre qui ne soit pas la ligne suivante. Pour cela il faut lui indiquer un repère. Ce repère est un nombre entier écrit dans les cinq premières colonnes de la ligne fortran. C'est l'étiquette.

Si l'utilisateur ne répond pas correctement à un ordre READ(∗,∗), au lieu de laisser le programme s'interrompre, on peut l'orienter vers une nouvelle exécution en utilisant le mot_clef **ERR** (éventuellement vers l'écriture d'un message explicite):
En cas d'erreur sur N et M le programme effectue l'instruction 10; dans le cas normal il continue sur l'instruction suivante.

```
^10^^^WRITE(*,*) 'ENTRER DEUX NOMBRES SEPARES PAR UNE VIRGULE'
^^^^^^READ(*,*,ERR=10) N,M
```

De même si aucune valeur n'est envoyée (retour chariot seulement), on peut orienter le programme vers une nouvelle exécution en utilisant le mot_clef **END** (éventuellement vers l'écriture d'un message explicite):

```
^^^^^^READ(*,*,END=10) N,M
```

Cette instruction est surtout utile lorsqu'on lit les données dans des fichiers tout préparés au lieu de les entrer en utilisant le clavier.

III.2) Variables booléennes.

Il s'agit d'une variable qui ne peut prendre que deux valeurs, **.TRUE.** (vrai) ou **.FALSE.** (faux). Cette variable doit avoir été déclarée par:

```
^^^^^^LOGICAL^P,N
```

... suite du programme ...

```
^^^^^^P=.TRUE.
^^^^^^READ(*,*) N
```

La donnée N peut être constituée uniquement de T ou F (les points ne sont pas obligatoires).

Une variable booléenne est associée à une proposition qui est vraie ou fausse selon que vous l'avez déclarée telle. Il s'agit soit d'une donnée que vous avez introduite (pour contrôler l'impression de résultats intermédiaires d'un calcul par exemple), soit de variables qui ont pris des valeurs selon des cas de figure prévues dans votre programme. Dans ce dernier cas, elle résulte de la comparaison de variables du programme à un instant donné; cette comparaison se fait au moyen d'opérateurs relationnels.

III.3) Opérateurs Logiques

Ces opérateurs agissent sur les variables booléennes.

.NOT. négation
`.NOT..TRUE.` équivaut à `.FALSE.`
`.NOT.P` équivaut à `.FALSE.` si `P=.TRUE.`

.AND. ajoute deux variables booléennes, la proposition est vraie si les deux propositions sont vraies.

.OR. la proposition résultante est vraie si l'une des deux propositions est vraie.

.EQV. la proposition résultante est vraie si les deux propositions sont équivalentes (toutes deux vraies, ou toutes deux fausses).

.NEQV. la proposition résultante est vraie si les deux propositions ne sont pas équivalentes (l'une vraie et l'autre fausse).

La hiérarchie de ces opérateurs est l'ordre ci-dessus (entre EQV et NEQV, c'est le premier rencontré, le plus à gauche)

III.4) Opérateurs relationnels

Ils permettent de comparer des variables non logiques de même type et de même précision (deux entiers, deux réels, deux chaînes de caractères...) Le résultat de la comparaison est **.TRUE.** (vrai) ou **.FALSE.** (faux).

.GT.	plus grand que	**>**
.GE.	plus grand que	**≥**
.LT.	plus petit que	**<**
.LE.	plus petit que	**≤**
.EQ.	égal à	**=**
.NE.	différent de	**≠**

Ces opérateurs sont prioritaires sur les opérateurs logiques.
Attention aux tests d'égalité sur les réels. Ceux-ci peuvent avoir des valeurs approchées et non rigoureusement égales. Pour que deux réels soient égaux, il faut que tous leurs bits le soient.

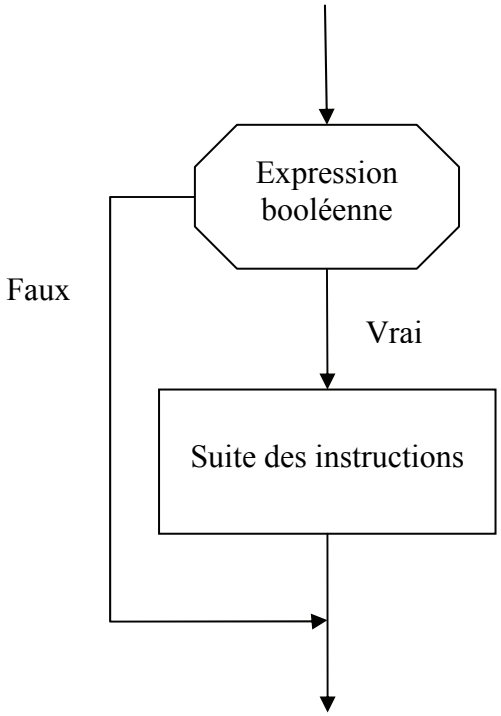
III.5) Ruptures de séquence : les Aiguillages.

Si la condition ne porte que sur une seule instruction :

```
IF ( <<expression booléenne>> )^<<instruction fortran>>
```

L'instruction fortran n'est exécutée seulement si l'expression booléenne (entre parenthèses) est vraie

Si la condition porte sur plusieurs instructions fortran :



l*a lecture du programme est facilitée si vous décalez les écritures des instructions vers la droite (colonne supérieure à 7)*

```
IF ( <<expression booléenne>> ) THEN

      … suite d'instructions à exécuter
      si l'expression est vraie …

ENDIF
```

Dans le cas où il y a plusieurs suites d'instructions suivant la justesse de l'expression booléenne :

```
IF ( <<expression booléenne>> ) THEN

      … suite d'instructions à exécuter
      si l'expression est vraie …
```

```
ELSE

        … suite d'instructions à exécuter
        l'expression est fausse …

ENDIF
```

Attention, il est nécessaire de respecter les sauts de lignes !

Il peut ne pas y avoir de ELSE (Si la proposition est fausse, les instructions du bloc suivant le IF..THEN sont omises). Il est possible d'imbriquer plusieurs blocs IF..THEN (Un second bloc IF..THEN-ELSE-ENDIF est alors inclus dans une série d'instructions contrôlée par un IF..THEN; dans ce cas, il y a autant d'ENDIF que de IF..THEN et chaque ENDIF renvoie toujours au dernier IF THEN rencontré.

```
IF ( <<expression booléenne 1>> ) THEN

        … suite d'instructions à exécuter si l'expression 1 est vraie …

ELSE
      IF ( <<expression booléenne 2>> ) THEN

              … suite d'instructions à exécuter si l'expression 1
              est fausse et l'expression 2 est vraie …

      ELSE

              … suite d'instructions à exécuter
              si les expressions 1 et 2 sont fausses …

      ENDIF    [sur l'expression booléenne 2]
ENDIF    [sur l'expression booléenne 1]
```

L'instruction **ELSE IF () THEN** *(sur une seule ligne)* permet d'alléger l'écriture de plusieurs propositions mutuellement exclusives. L'ensemble commandé par le premier IF..THEN se termine alors par un seul ENDIF (pas de ENDIF associés aux ELSE IF sur une seule ligne).

```
IF ( <<expression booléenne 1>> ) THEN

        … suite d'instructions à exécuter si l'expression 1 est vrai …

ELSE IF ( <<expression booléenne 2>> ) THEN

        … suite d'instructions à exécuter
        si l'expression 1 est fausse et l'expression 2 est vraie …

        ELSE

        … suite d'instructions à exécuter si les expressions 1 et 2 sont fausses …

ENDIF
```

Attention:

• *Une seule instruction par ligne; n'ajoutez rien aux lignes où sont déjà écrits if..then ou else.*

• *Il y a autant d'ENDIF que de IF..THEN (Ne pas compter les ELSE IF..THEN)*

• *Le nombre d'ELSE est au plus égal à celui de IF..THEN*

• *Le nombre d'ELSE IF..THEN est quelconque (Les propositions sont alors hiérarchisées - les instructions qui suivent un ELSE IF..THEN sont effectuées dans le cas où toutes les propositions booléennes qui précèdent sont "false").*

Le programme suivant ne contient qu'un seul endif.

```
      PROGRAM TEST
      IMPLICIT NONE
      READ(*,*) I
      IF (I.LT.2) THEN
        WRITE(*,*) I,' est < 2'
      ELSE IF (I.EQ.3) THEN
        WRITE(*,*) I,' vaut 3'
      ELSE IF(I.LT.10) THEN
        WRITE(*,*) I,' est compris entre 4 et 10'
      ENDIF
      STOP
      END
```

Evitez d'écrire au départ beaucoup d'if sans les tester; éviter les else if si vous débutez.

III.6) Flags et Arrêts d'un programme.

Un programme fortran doit se terminer par END

L'instruction **STOP** dans le programme l'arrête.

L'instruction **CONTINUE** dans le programme fait passer à l'instruction suivante. Elle est utile pour porter une étiquette (certaines instructions n'autorisent pas l'écriture d'une étiquette).

L'instruction **PAUSE N** dans le programme le suspend (N entier ou réel selon les versions fortran).

Un **Flag** est une variable lue en donnée qui détermine une rupture de séquence. Vous pouvez dans un programme à options faire choisir par l'utilisateur quel branchement suivre ou simplement lui demander en cas de boucle, s'il faut continuer ou arrêter. Faites toujours précéder la lecture du Flag par des écritures claires sur l'écran (proposez un menu complet en précisant les réponses demandées - prévoyez aussi que la réponse ne soit pas du type demandé en utilisant le mot-clef ERR).

Exercices:

III-1) Ecrire un programme qui permet de rentrer le nom, l'âge, la taille et le sexe d'une personne et qui imprime à partir de ces renseignements que la personne est majeure (≥18 ans) ou mineure, grande moyenne ou petite.
On considéra que la taille moyenne d'une femme est comprise entre 1.5 et 1.7 mètres et que celle d'un homme est comprise entre 1.6 et 1.8 mètres.
III-2) Résolution de l'équation du second degré AX^2+BX+C (faire lire A, B et C; Prévoir tous les cas possibles. Ne pas déclarer de variable complexe). Essayer de programmer avec le minimum d'étiquette.

III-3) Recherche de la racine d'une fonction F(x) par dichotomie:

Le principe est de réduire l'écart (x_a,x_b) entre deux valeurs de x conduisant à des valeurs F(x) opposées. Si l'on a trouvé deux valeurs initiales, $F(x_a)$ et $F(x_b)$ de signe opposé, on calcule $F(\frac{x_a+x_b}{2})$ qui est de signe contraire soit de x_a, soit de x_b. La racine est donc comprise dans l'intervalle correspondant aux deux valeurs contraires. On a ainsi divisé par deux l'intervalle (x_a,x_b). En procédant par itération, on obtient un intervalle aussi réduit que l'on veut.

***Ecrire l'organigramme**. Réaliser le programme en prévoyant que les premières valeurs (x_a,x_b) ne soient pas forcément satisfaisantes. Tester le programme pour F(x)=sinx+2x-2*

Insérer une instruction
*F(R)=sin(R)+2*R-2*
après les déclarations et utiliser F() comme si c'était une fonction bibliothèque.

III-4) Calculer le rapport du volume d'une sphère de rayon a par rapport au volume du cube d'arête 2a, en tirant aléatoirement des points dans le cube, en cherchant s'ils sont situés à l'intérieur d'une sphère et en sommant le nombre de réponses positives. En déduire la valeur du nombre π en fonction du nombre de points tirés.

Pour réaliser ce programme, il est nécessaire d'utiliser un générateur de points aléatoires.

Le générateur RAND() donne une liste de valeurs dites aléatoires comprises entre 0. et 1. Cette liste est toujours engendrée de la même façon et donc est toujours la même. Elle est constituée de façon à obtenir une distribution homogène dans un intervalle de nombres.

La variable RAND() va alors contenir un réel compris entre 0 et 1.
Pour obtenir une variable réelle E comprise entre 0 et 1 écrire
E=RAND()