Syntaxe et règles operatoires

1 Introduction au Fortran

- langage utilise principalement pour le calcul scientifique.
  standard
- Variables en minuscule, pas d'accents
- typer toutes les variables (implicit none)
  (a-h,o-z) real , (i-m) integer
- Format fixe : instructions entre colonnes 7 et 72
  colonne 6 prolongation ligne precedente, colonne 1 commentaire (c)
- L'espace ne joue pas de role, utilisez le pour que votre code soit lisible: indentez les boucles, 'aerez' vos formules...
- fortran passe des adresses, toutes les variables sont locales
  pour un tableau fortran passe l'adresse du premier element
  C passe des valeurs
- pas de melanges de types de flottants, ou real ou double precision pas les deux

2 Evaluation des expressions

Le resultat d'operations depend du type de variables utilisees. Lorsque l'on ecrit n3 =n1/n2 ou n1,n2 et n3 sont des entiers, le resultat est le quotient de la division euclidienne de n1 par n2.

Une operation entre deux variables donne un resultat qui est represente par la machine par le type le plus eleve des deux variables en respectant la hierarchie

double precision > real > integer

exemples:
$1./2 \rightarrow 0.5$
$1/2 \rightarrow 0$

Les priorités des operateurs arithmetiques sont les suivantes
$** > */ > +-$

Les priorités des operateurs logiques sont les suivantes
1) comparateurs .gt. .ge. .eq. .ne. .le. .lt.
2) .not.
3) .or.
4) .eqv. .neqv.

Les priorités des operateurs sont les suivantes
1) operateurs arithmetiques
2) concatenation
3) comparaison
4) operateurs logiques

Dans **l'evaluation d'expressions complexes** le compilateur suit l'ordre suivant
1) appels de fonctions
2) expressions entre parentheses en commmencant par les plus internes
3) le reste est evalué en suivant les priorités des opérateurs de gauche à droite pour les opérateurs de priorités égales

3 Boucle a nombre fixe d'iterations

Repetition d'un grand nombre d'operations en variant des données

notation recente (fortran 90 et 95) , plus claire

```
      do i =n1,n2,n3
.....

      enddo
```

avec les regles suivantes:
n1 = n2 la boucle est effectuee une fois
n1 > n2 et n3 >0 , la boucle n'est pas effectuee
n1 < n2 et n3 >0 , la boucle est effectuee pour i=n1,n1+n3 n1+2*n3,...n1+k*n3 avec k tel que n1+(k+1)*n3 > n2

Dans l'exemple donne plus haut n3=1 (la valeur par defaut)

4 Boucle conditionnelle

- Repetition d'un grand nombre d'operations jusqu'a ce qu'un resultat de test change.
- do while (tant que) obsolete
  remplace par boucle do avec l'instruction exit de sortie de boucle
  permet un meilleur controle de flux.

```
do

  if (condition) then
    exit
  endif

enddo
```

Exemple: iteration $x_{n+1} = f(x_n)$ avec comme criteres d'arret $n < nmax$ ou $|x_{n+1} - x_n| < tol$

Version avec exit

```
x = x0
n = 0
res = abs(x0 - f(x0))
icon = 1
do n =1,nmax
     x1 = f(x)
     res = abs(x1-x)
     if (res < tol) then
        icon = 0
        exit
     endif
     write(*,*) n , x1 , res
     x= x1
enddo
```

L'indice icon indique quel est la raison pour l'arret
$icon = 0$ convergence , sinon nmax iterations sans converger Version moins precise avec le do while (ne permet pas de preciser la raison de l'arret)

```
x = x0
n = 0
res = abs(x0 - f(x0))
do while ((n < nmax) .and. (res > tol)
     n = n+1
     x1 = f(x)
     res = abs(x1-x)
     write(*,*) n , x1 , res
     x= x1
enddo
```

Remarques: Apres la sortie de boucle la valeur de l'indice de boucle est perdue donc il ne faut jamais l'utiliser hors de la boucle.
Un indice de boucle ne doit jamais etre modifie dans la boucle.