FORTRAN se caractérise par la nécessité de compiler les scripts, c'est à dire transformer du texte en binaire.(transforme un fichier de texte en .f95 en un executable (non lisible par un éditeur) en .exe.)
Avant de commencer, tout ce qui est écrit en bleu désigne du vrai langage FORTRAN, ce qui est en *italique* désigne un truc général et une ligne débutant par >> désigne du SHELL.

1. Structure d'un programme FORTRAN 95

D'abord, ouvrir un éditeur sous LINUX. ex. nedit ou gedit.
>> nedit &
(Le & permet d'ouvrir l'éditeur tout en laissant le terminal utilisable pour d'autres commandes)

Un script FORTRAN se décompose en:

program *nom du programme*
Déclaration des variables
instructions
éventuellement des comentaires, précédés d'un !
end program *nom du programme*

OK, FORTRAN est un peu neuneu, alors il faut lui dire clairement au début du programme le nom et le type des variables que l'on va utiliser.
Avant de déclarer les variables, entrer implicit none, sinon FORTRAN attribuera automatiquement un entier aux variables nommées n, i, j, k...
Puis déclarer. Par exemple, pour déclarer l'entier a, le caractère b, les réels a, toto et titi, entrer respectivement:

```
integer::a
character::b
real::a,toto,titi
```

Viennent ensuite les instructions:

```
a=10
toto=2.1
titi=a+toto
```

Et toutes les opérations classiques: + - * / = et ** qui signifie puissance.

Commande d'écriture

Pour « imprimer un truc à l'écran », utiliser la commande print:

```
print*, titi
```

Et la valeur de la variable titi (12.1) s'affichera à l'écran.
Mais si l'on veut écrire bonjour, print*, bonjour ne donnera rien, car bonjour n'est pas une variable déclarée... Pour afficher le mot bonjour, il faut taper:

```
print*, 'bonjour'    (' ou « )
```

Commande de lecture

Pour que le programme « lise » une valeur ou des caractères avant de les utiliser, employer la commande read:

Pour demander à l'utilisateur d'entrer une valeur pour a, et l'intégrer au programme, il suffit de:

```
print*, « entrer la valeur de a »
read*,a
```

Pour compiler son programme (test.f95, par exemple), lancer dans le terminal LINUX:
>> gfortran -o test.exe test.f95
Ceci crée un fichier exécutable: test.exe. Pour le lancer, taper
>> ./test.exe

2. Tests, Instructions conditionnelles et Boucles

Les instructions conditionnelles (if, then, else, en gros.)

```
if (test 1) then
instructions
else if (test 2) then
instructions
else
instructions
end if
```

Les tests...
On démarre avec les « opérateurs relationnels »

>	ou	.GT.	en FORTRAN 77
<		.LT.	
>=		.GE.	
<=		.LE.	
==		.EQ.	(égal)
/=		.NE.	(non égal)

Un petit exemple pour la route: un programme qui donne le maximum entre deux réels:

```
program max
implicit none
real::a,b
print*,'donner les 2 valeurs'
read*,a,b
if (a<b) then
print*,b
else
print*,a
end if
end program max
```

Idem avec les « opérateurs logiques »
Car il existe aussi des variables logiques, qui prennent les valeurs .TRUE. et .FALSE.
On peut même les déclarer...

```
integer::a,toto
LOGICAL::test1
test1=a<1
print*,test1
```

Ceci affichera T (true) si a est effectivement inférieur à 1, et F (false), sinon.
On peut alors continuer en se contentant d'écrire:

```
if (test1)
toto=4
else
toto=-2
endif
```

Et les opérateurs logiques ?
ex. si on doit écrire « si le test 1 et vrai et si le test 2 est vrai aussi, alors... »
On doit définir:

.AND.	(et)
.OR.	(ou inclusif)
.NOT.	(et non)

Un p'tit exemple:

```
if ((test1) .AND. (b<=5.6)) then
...
```

Et pour s'éviter pas mal de boucles if, on se simplifie la vie avec select case

```
select case (variable)
case (liste de valeurs qu'elle peut prendre)
instructions
case (une autre liste de valeurs possibles)
instructions
case default
instructions
end select
```

Un petit exemple de programme: money.f95, qui convertit les euros en dollars ou en francs, et inversement, dans tous les sens souhaités:

```
!convertisseur euros-franc-dollars (cours au 28/2/08)
program money
implicit none
character::dev1
character::dev2
real::s
print*,'entrer la somme, sa devise et la devise souhaitée (f,e,$)'
read*,s,dev1,dev2
if (dev1=='$') then
      if (dev2=='e') then
      print*,s/1.50
      elseif (dev2=='f') then
      print*,(s/1.50)*6.55957
      endif
elseif (dev1=='e') then
      if (dev2=='f') then
      print*,s*6.55957
      elseif (dev2=='$') then
      print*,s*1.50
      endif
elseif (dev1=='f') then
      if (dev2=='e') then
      print*,s/6.55957
      elseif (dev2=='$') then
      print*,(s/6.55957)*1,50
      endif
endif
end program money
```

Les boucles.

Il y en a deux types: les boucles do, lorsque l'on connait à l'avance le nombre d'itérations à effectuer, et les boucles while, lorsqu'on veut foncer, à l'arrache.

Par exemple, calculons la somme de $1/i^2$, pour i allant de 1 à 100:

```
real::sum
integer::i
sum=0
do i=1,100,1
sum=sum+1/(i*i)
enddo
```

Autre exemple, cherchons à partir de quel n la somme précédente de 1 à n devient supérieure à 1.5:

```
sum=0
i=0
do while (sum<1.5)
i=i+1
sum=sum+1/(i*i)
enddo
```

Le (i-1) final nous donnera accès à n...

ATTENTION, si on laisse tourner des boucles à l'infini... (ex un while avec un test toujours vrai...)
Pour casser la boucle, taper
>> Ctrl C

3. Entrées et Sorties

Jusqu'ici, on a printé et readé des données que l'utilisateur doit fournir lui même, en direct, à la main. Mais il est souvent pratique de lire des valeurs en entrée dans un fichier, et aussi d'écrire les résultats en sortie dans un fichier... Dans les deux cas, il y a 3 étapes:

Ouvrir le fichier (open)

Le fichier doit être placé dans le même répertoire que le script. Il faut lui assigner un numéro, et préciser si c'est un fichier en binaire (unformatted) ou texte (formatted). Pour ouvrir le fichier texte toto.txt, entrer:

```
open(unit=1,file= « toto »,form= « formatted »,action= « read »,iostat=ios)
```

Trifouiller le fichier (read ou write)

Pour lire le fichier, il faut d'abord le décrire. Contient-il des entiers, des réels ou des caractères (respectivement I, F et A) ?
Combien de caractères faut-il lire pour chaque type ?
Exemple: pour lire un fichier en colonnes listant des données sismiques (stations.txt, avec station, latitude, longitude et altitude de la station):

```
N    28.00160   84.62350   1997
```

Si l'on veut lire la colonne de latitude (nombre réel), on veut identifier 8 caractères, dont 5 après la virgule. Il faut alors écrire F8.5

Et pour tout lire:

```
character::sta
real::lat,lon
integer::alt
open(unit=1,file= « stations.txt »,form= « formatted »,action= « read »)
read(unit=1,fmt='(A,F8.5,F8.5,I5)') sta, lat, lon, alt
```

FORTRAN lira le fichier de façon séquentielle, ligne par ligne.

Pour écrire des données dans un fichier, le principe est le même, il faut préciser le format dans lequel on va écrire les variables:

```
write(unit=1,fmt='(  )') variables
```

Exemple: si i=12 et j=1234:

```
integer::i,j
open(...
write(unit=1,fmt='(I5,I8)') i, j
```

va écrire: ---12----1234

Par ailleurs,

```
write(unit=1,fmt='(« i= »,I5, « j= »,I8)') i, j
```

va écrire: i=---12j=----1234, ce qui peut clarifier les choses...

Enfin, surtout, ne pas oublier de fermer le fichier avec close :

```
close(unit=1)
```

4. Notions sur les tableaux, Chaînes de Caractères

Un tableau est une variable pouvant stocker plusieurs valeurs **de même type** (entier, réel, caractère...). Par exemple, si l'on déclare le tableau suivant comme un tableau d'entiers:

4	-2	6	1	5	8	686	1.797

Il y aura un bug à cause de la dernière valeur, réelle.
FORTRAN manipule des tableaux jusqu'à 7 dimensions (une matrice étant un tableau à deux dimensions).

Comment les déclarer proprement ?

type ,dimension(*expression1*,*expression2*,...,*expression n*):: *nom du tableau*

Déclarons par exemple le tableau suivant, nommé titi (1X6), de trois façons différentes, pour le fun:

4	-1	2	6	3	1

```
integer, dimension(6)::titi
integer, dimension(1:6)::titi
integer, dimension(-1:4)::titi
integer, dimension(0:5)::titi
```

Par défaut, le premier indice est 1, mais on peut lui attribuer 0.
Si le tableau fait plus d'une dimension, on sépare les dimensions par des virgules:

```
real, dimension(4,6)::toto
```

pour déclarer:

65.9	54.5	45.5	43.35	42.24	212.3
64.98	35.0	5.5	23.2	78.9	323.3
35.98	13.4	545.3	43.6	34.5	44.4
32.34	43.6	34.5	53.2	324.34	20.08

Un peu de vocabulaire:
Le **rang** d'un tableau = nombre de dimensions (ex. = 2 pour toto, =1 pour titi)
Le **profil** d'un tableau = vecteur avec le nombre d'éléments dans chaque dimension (ex. [4,6] pour toto)
La **taille** d'un tableau = produit des éléments du profil (ex. 24 pour toto)
Deux tableaux sont **conformants** quand ils ont le même profil.

Pour travailler sur des morçeaux de tableaux, à partir de:

```
integer, dimension(4,6)::toto
```

`toto(2,3)` renvoie 5.5
`toto(:,:)` renvoie toto en entier
`toto(1:3,2)`
renvoie la portion:

54.5
35.0
13.4

On peut aussi extraire des données du tableau avec un pas choisi. Par exemple, pour isoler les éléments de la 1ere ligne de deux en deux, entrer:

`toto(1,2:6:2)` ce qui sortira :

54.5	43.35	212.3

Et pour effectuer des opérations sur les tableaux ?
FORTRAN effectue toutes les opérations membre à membre (et non des produits matriciels, directement, par exemple).

Remplir les tableaux (affectation, avec =):

Pour attribuer la valeur 1 à la case (2eme ligne, 3eme colonne):

```
toto(2,3)=1
```

Remarque... `toto=1` est autorisé, bien que toto et 1 ne soient pas conformants. Cette commande remplit toto de « 1 », dans toutes les cases.

Opérations diverses...
Tout est permis, mais terme à terme: + - / * sin cos ...

Et les chaînes de caractères...

Rappel: pour déclarer un caractère, il suffit d'entrer:

```
character::ch
```

Mais pour une chaîne de caractères:

`character(LEN=` *longueur de la chaîne*`)::` *nom de la chaîne*

Par exemple:

character(LEN=9):: ch

déclare ch comme un tableau de caractères à 1 ligne et 9 colonnes.

Mais, grande différence avec les tableaux, si on attribue

ch= « bonjour »

La commande va fonctionner, même si « bonjour » n'a que 7 lettres:

B	O	N	J	O	U	R		

Des espaces seront automatiquement attribués aux cases non remplies.

A part ça, on peut réaliser les mêmes opérations que sur les tableaux, par exemple transformer « bonjour » en « bonsoir », en attibuant:

ch(4:7)= « soir »

Remarque chiante: pour désigner le i-ème caractère d'une chaîne de caractères, ch(i) ne marche pas. Il faut entrer ch(i:i) ...

5. Les Procédures

Comment faire appel à de nombreux petits programmes secondaires au sein d'un programme principal ?
Le programme principal doit faire **appel** aux procédures avec des **arguments** (variables).
On distingue deux types de procédures:

Une Subroutine:

C'est une procédure dite « non typée », c'est à dire qui ne se déclare pas / dont on ne déclare pas le type. Pour l'appeler, employer la fonction call...

call *nom de la subroutine* (*arguments*)

Exemple... écrire un programme qui demande d'écrire des valeurs dans un tableau et en affiche le maximum et la moyenne au fur et à mesure.

```
program meanmax
implicit none
real,dimension(100)::tab
real::moy,max
integer::i
do i=1,100
print*, « entrer la nouvelle valeur »
read*,tab(i)
call mm(tab,i,100,moy,max)
! la subroutine mm permet de calculer max et moy
print*,max, moy
enddo
end program meanmax
```

La subroutine mm s'écrit alors (pas de program ni de end program)

```
subroutine mm(t,ind,n,moy,max)
! les noms des arguments de la subroutine
! peuvent être différents de ceux du programme principal
integer::ind,n,i
real::moy,max
real,dimension(n)::t
! dans une subroutine la dimension d'un tableau peut être une variable
moy=0
max=t(1)
do i=1,ind
if (t(i)>max) then
max=t(i)
end if
moy=moy+t(i)
end do
moy=moy/ind
end subroutine
```

Une fonction:

C'est une procédure **typée**, donc à déclarer.
Par exemple, la fonction valeur absolue (v=abs(-36)). Elle est typée real, puisqu'elle admet en entrée un réel pour ressortir un réel.
Inutile d'appeler 1 fonction avec call, juste: *nom de la fonction (arguments)*

Par exemple, reprenons le programme meanmax.f95 avec des fonctions:
La déclaration est la même mais il faut en plus **déclarer la fonction que l'on va utiliser**.
On va créer une fonction pour donner le max d'un tableau (fmax) et une autre pour la moyenne (fmean).

```
Idem que précédent, déclaration...
real::fmax,fmean
do ...
read ...
max=fmax(t,i,100)
moy=fmean(t,i,100)
```

Ecrivons à présent fmax.

ATTENTION ! Connerie du FORTRAN (...) il faut **déclarer la fonction au sein même du script de la fonction !**

```
function fmax(t,ind,n)
integer::ind,n
real,dimension(100)::t
real::fmax
fmax=t(1)
do i=1,ind
if (t(i)>fmax) then
fmax=t(i)
end do
end function
```

Et enfin fmean.

```
function fmean(t,ind,n)
integer::ind,n
real,dimension(100)::t
real::fmean
fmean=0
do i=1,ind
fmean=fmean+t(i)
end do
fmean=fmean/ind
end function
```

Et enfin pour conclure, comment compiler tout ce bordel de programmes principaux, subroutines et autres fonctions ?

Il faut procéder en 2 étapes:
La compilation proprement dite, puis l'édition de liens (loader) entre les différents programmes.

1/ La compilation indiquera les éventuelles erreurs, traduira le texte en langage machine, et va créer des objets en « .o » dits « modules ». La commande correspondante est
>> gfortran -c *tous les programmes .f95 à compiler*

2/ L'édition de liens va relier les fichiers entre eux et fabriquer un executable (en « .exe »). Pour cela, taper:
>> gfortran -o *nom souhaité pour l'executable* .exe *tous les modules créés précédemment* .o

Exemple pour le programme meanmax.f95 avec les fonctions fmean.f95 et fmax.f95 :
>> gfortran -c meanmax.f95 fmax.f95 fmean.f95
>> gfortran -o meanmax.exe meanmax.o fmax.o fmean.o