INP TOULOUSE ENSEEIHT

# Programmation impérative: Méthode de Programmation

# 3. Fonctions + Subroutines

# Table des matières


- Functions 8
- Subroutines 20

# Qu'est qu'une procédure?

Une procédure résulte du raffinement d'une action abstraite que le processeur n'est pas en mesure d'exécuter.

Une fois définie, elle devient une action élémentaire ou une "instruction" disponible pour le processeur

Une procédure manipule **des variables**:

- soit **locales** à la procédure, c'est-à-dire qu'elle seule les connait et les utilise
- soit **proviennent de l'extérieur**, c'est-à-dire qu'elles sont transmises à la procédure (passage d'arguments)

# Pourquoi des procédures (sous-programmes, fonctions) ?

**Compréhension globale du programme**

Lecture du programme principal plus compréhensible (le nom de la procédure donne l'intention)

**Compréhension détaillée du programme**

Procédures « courtes » = facile à comprendre

**Factorisation**

Plus de copier-coller = moins d'erreurs

**Mise au point facilitée**

Tests faits à l'échelle d'une procédure = débuggage plus rapide

# Ex.: Le diabolo menthe / fraise

## Programme principal

**Algorithme Niveau 1**

```
Variables : Verre, sirop de fraise, sirop de
menthe, glaçon, limonade

Début Algorithme

Prendre un verre
SP Mettre du sirop dans le verre avec …
Mettre de la limonade dans le verre
Mettre un glaçon dans le verre

Fin Algorithme
```

# Ex.: Le diabolo menthe / fraise

## Programme principal

**Algorithme Niveau 1**

```
Variables : Verre, sirop de fraise, sirop de
menthe, glaçon, limonade

Début Algorithme

Prendre un verre
SP Mettre du sirop dans le verre avec …
Mettre de la limonade dans le verre
Mettre un glaçon dans le verre

Fin Algorithme
```

## Sous-programme

**Algorithme SP Mettre Sirop dans le verre**

```
Variables : sirop de fraise(IN), sirop de
menthe(IN), verre(IN/OUT), choix (local)

Début Algorithme
Demander choix
Si choix Menthe
  Alors Mettre du sirop de Menthe dans le verre
Sinon Si choix Fraise
  Alors Mettre du sirop de fraise dans le verre
FinSi

Fin Algorithme
```

# Ex.: Le diabolo menthe / fraise

Programme principal

**Algorithme Niveau 1**

```
Variables : Verre, sirop de fraise, sirop de
menthe, glaçon, limonade

Début Algorithme

Prendre un verre
SP Mettre du sirop dans le verre avec …
Mettre de la limonade dans le verre
Mettre un glaçon dans le verre

Fin Algorithme
```

Sous-programme

**Algorithme SP Mettre Sirop dans le verre**

```
Variables : sirop de fraise(IN), sirop de
menthe(IN), verre(IN/OUT), choix (local)

Début Algorithme
Demander choix
Si choix Menthe
  Alors Mettre du sirop de Menthe dans le verre
Sinon Si choix Fraise
  Alors Mettre du sirop de fraise dans le verre
FinSi

Fin Algorithme
```

**Les variables dont le sous-programme a besoin pour fonctionner sont :**

- Le sirop de fraise ➔ on ne veut pas modifier le sirop de fraise juste y accéder (IN)
- Le sirop de menthe ➔ on ne veut pas modifier le sirop de menthe juste y accéder (IN)
- Le verre ➔ on veut « modifier » le verre pour y mettre du sirop (IN-OUT)
- choix est une variable locale utilisées seulement par le sous-programme

# FUNCTION

Un peu comme les « fonctions mathématiques »

Fournit un seul résultat

# Les fonctions

## Algorithmique

Définition de la fonction f telle que:

$f(a,b,x) = (a\ x^3 + b/x\ ) /2$

Entrée: a, b, x

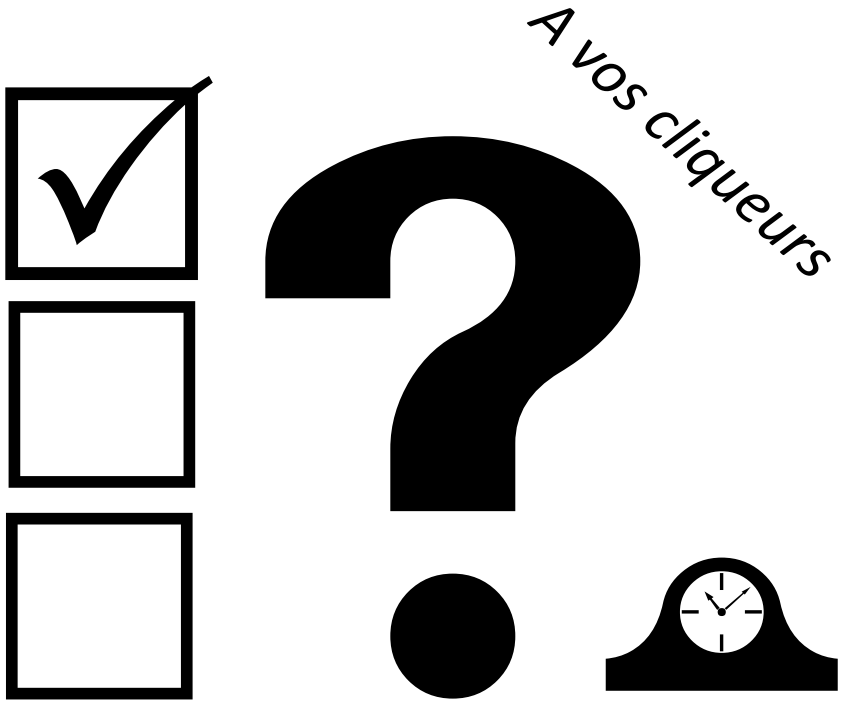
Résultat: f

Nota: le symbole « f » ne pourra plus être utilisé ailleurs, il est affecté à la fonction

## FORTRAN

```
! Definition de f
FUNCTION f(a,b,x)
   IMPLICIT NONE
   REAL, INTENT (IN) :: a,b,x
   REAL :: f
   f = ( a * x**3 + b / x ) / 2.
END FUNCTION f

! Programme principal
program TOTO
implicit none

REAL :: a,b,x
REAL :: f

print*,"Donner les valeurs de a, b, x"
read*, a,b,x
print*,"f(a,b,x)=",f(a,b,x)

end program TOTO
```

arguments de la fonction


A vos cliqueurs

# Qu'affiche le programme quand l'utilisateur entre a=5 b=3 x=1 ?

Quiz

A. f(a,b,x)=f(5,3,1)

B. 4

C. f(a,b,x)=4

D. f(5,3,1)=4

E. ni A, ni B, ni C, ni D…

FORTRAN

```
! Definition de f

FUNCTION f(a,b,x)
   IMPLICIT NONE
   REAL, INTENT (IN) :: a,b,x
   REAL :: f
   f = ( a * x**3 + b / x ) / 2.
END FUNCTION f

! Programme principal
program TOTO
implicit none

REAL :: a,b,x
REAL :: f

print*,"Donner les valeurs de a, b, x"
read*, a,b,x
print*,"f(a,b,x)=",f(a,b,x)

end program TOTO
```

# Exemple

- On veut écrire une fonction f fournissant en résultat la valeur de $(a\ x^3 + b/x\ ) /2$, les valeurs de a, b et x étant fournies en argument.
- On veut écrire un programme qui affiche les valeurs de la fonction f pour x=1, 2, 3, …, 10.

## FORTRAN

```
! Procédure
FUNCTION f(u,v,w)
   IMPLICIT NONE
   REAL, INTENT (IN) :: u,v,w
   REAL :: f
   f = ( u * w**3 + v / w ) / 2.
END FUNCTION f
```

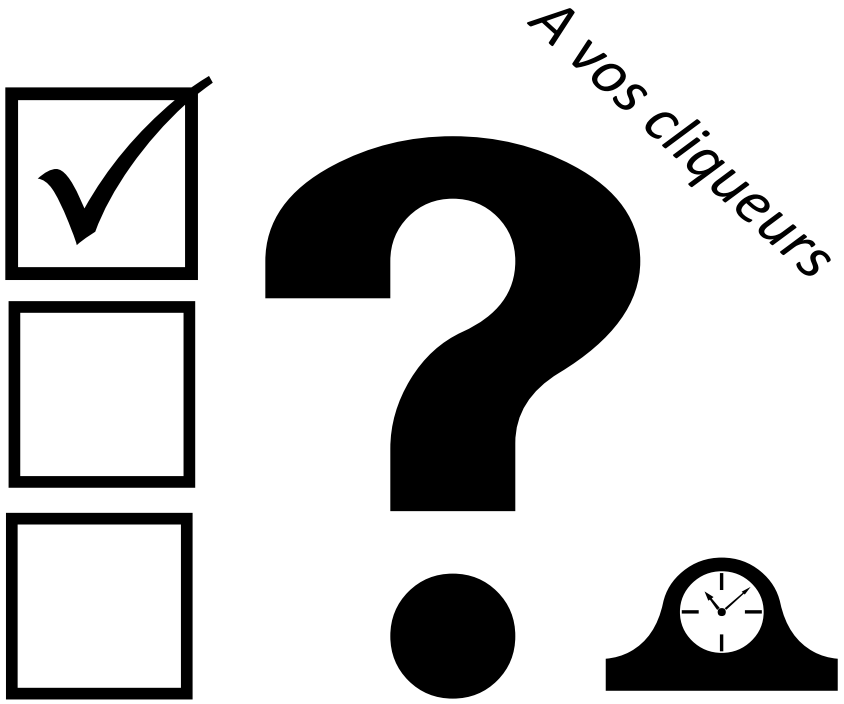
## FORTRAN

```
! Programme
PROGRAM valeur_f
   IMPLICIT NONE
   REAL :: a, b, x, res
   REAL :: f
   INTEGER :: i
   a = 5.
   b = 3.
   DO i = 1, 10
      x = real(i)
      res = f(a, b, x)
      PRINT*,'Pour x= ', x, 'le resultat est ', res
   END DO
END PROGRAM valeur_f
```

# Exemple

## Lancement du programme - affichage à l'écran

```
Pour x =    1.0000000    le resultat est:   4.0000000
Pour x =    2.0000000    le resultat est:   20.750000
Pour x =    3.0000000    le resultat est:   68.000000
Pour x =    4.0000000    le resultat est:   160.37500
Pour x =    5.0000000    le resultat est:   312.79999
Pour x =    6.0000000    le resultat est:   540.25000
Pour x =    7.0000000    le resultat est:   857.71429
Pour x =    8.0000000    le resultat est:   1280.1875
Pour x =    9.0000000    le resultat est:   1822.6666
Pour x =    10.000000    le resultat est:   2500.1499
```


A vos cliqueurs

# Un autre exemple

Quiz

- Quel type de résultat donne la fonction ?
- Que fait la fonction ?
- Que fait le programme pour age=15 ?
- Que fait le programme pour age=19 ?

```
! Function
FUNCTION test_majorite(toto)

   IMPLICIT NONE
   REAL, INTENT (IN) :: toto
   LOGICAL :: test_majorite

   IF ( toto >= 18. ) THEN
       test_majorite = .TRUE.
   ELSE
        test_majorite = .FALSE.
   END IF

END FUNCTION test_majorite

! Programme
PROGRAM entree_disco

   IMPLICIT NONE
   REAL :: age
   LOGICAL :: test_majorite

   PRINT*, 'Donner son age'
   READ*, age
   IF (test_majorite(age) .eqv. .TRUE.) THEN
       PRINT*, 'Bonne soiree'
   ELSE
        PRINT*, 'Revenez dans' , 18 - age , 'ans'
   END IF

END PROGRAM entree_disco
```

# Fonctions intrinsèques de Fortran 90

## FORTRAN (liste non-hexaustive)

**! Fonctions mathématiques usuelles**

cos(a), acos(a), sin(a), asin(a), tan(a), atan(a), exp(a), log(a), log10(a), sqrt(a)

**! Fonctions numériques**

| | |
|---|---|
| abs(a) | ! Fournit la valeur absolue de a (entier, réel) |
| ceiling(a) | ! Fournit l'entier immédiatement supérieur à la valeur du réel a |
| floor(a) | ! Fournit l'entier immédiatement inférieur à la valeur du réel a |
| int(a) | ! Convertit a en entier |
| real(a) | ! Convertit a en réel |
| max(a1, a2, …, an) | ! Fournit la valeur max des ai (entiers ou réels) |
| min(a1, a2, …, an) | ! Fournit la valeur min des ai (entiers ou réels) |
| mod(a,p) | ! Fournit a-int(a/p)*p, le reste de la division euclidienne de a/p |
| random_number(a) | ! Fournit dans a un nombre pseudo-aléatoire tel que $0 \leq a < 1$ |

# Fonctions intrinsèques de Fortran 90

(suite)

```
! Fonctions relatives aux tableaux
dot_product(u,v)      ! Fournit le produit scalaire du vecteur u par le vecteur v
maxval(tab)           ! Fournit la valeur max des éléments du tableau tab
minval(tab)           ! Fournit la valeur min des éléments du tableau tab
sum(tab)              ! Fournit la valeur de la somme des éléments du tableau tab
maxloc(tab)           ! Fournit les indices de l'élément de valeur max du tableau tab
minloc(tab)           ! Fournit les indices de l'élément de valeur min du tableau tab
```

# Exercice

- Ecrire une fonction fournissant en résultat le volume d'une sphère dont le rayon (de type real) lui est fourni en argument.

- Ecrire le programme l'utilisant pour calculer 3 volumes correspondant à trois rayons fournis en donnée (au clavier)

## SUBROUTINE

= sous-programme avec plusieurs Entrées – Sorties

Ne fournit pas qu'un seul résultat

# SUBROUTINE

## FORTRAN

```
! Exemple de programme de facture
PROGRAM facture

  IMPLICIT NONE
  INTEGER :: n
  REAL :: pht, ptht, pttc

  WRITE(*,*) 'Donner n et le prix unitaire HT'
  READ(*,*) n, pht

  CALL prix(n, pht, ptht, pttc)

  WRITE(*,*) 'Le prix total HT est', ptht
  WRITE(*,*) 'Le prix total TTC est', pttc

END PROGRAM facture
```

Variables globales

## FORTRAN

```
SUBROUTINE prix( n,  pht,  ptht,  pttc )

  IMPLICIT NONE

  INTEGER, INTENT(IN) :: n
  REAL, INTENT(IN) :: pht
  REAL, INTENT(OUT) :: ptht, pttc
  REAL :: taxe

  taxe = 1.186
  ptht = real(n) * pht
  pttc = ptht * taxe

END SUBROUTINE prix
```

données

résultats

Variables en arguments

variable locale

# Genre des arguments

INTENT(IN) = la valeur peut être utilisée mais pas modifiée dans la procédure

INTENT(OUT) = la valeur peut être modifiée mais pas utilisée dans la procédure

INTENT(INOUT) = la valeur peut être utilisée et modifiée dans la procédure

En général, les variables de données sont du genre INTENT(IN)
les variables résultats sont du genre INTENT(OUT)

Exception: les variables modifiées **avant** la procédure et **dans** la procédure sont du genre INTENT(INOUT)

# Exemple du tri (4)

## Tri

Fonctionnalité : On veut trier par ordre croissant les éléments d'un vecteur contenant n éléments (n<=100), stockés dans un fichier texte nommé don.dat et écrire le résultats dans un fichier texte appelé res.dat.

On utilise pour cela:

- une fonction qui détermine la valeur maximum du vecteur

- plusieurs sous-programmes qui, respectivement:
  - -> lit les valeurs du vecteur non-trié dans don.dat
  - -> détermine la valeur et l'emplacement du minimum du vecteur
  - -> écrit les valeurs du vecteur trié dans res.dat

# Exemple du tri (4)

```
PROGRAM TRI
   IMPLICIT NONE
   ! DECLARATIONS
   INTEGER :: n
   REAL, DIMENSION(100) :: u, v

   ! INSTRUCTIONS
   CALL lecture_valeur('don.dat',n)
   CALL lecture_vecteur('don.dat',n,u)
   CALL tri_vecteur(n,u,v)
   CALL ecriture('res.dat',n,v)

END PROGRAM TRI

!-----------------------------------
SUBROUTINE lecture_valeur(nomfich,n)

   IMPLICIT NONE
   ! DECLARATIONS
   ! variables en arguments
   CHARACTER(len=7), INTENT(IN) :: nomfich
   INTEGER, INTENT(OUT) :: n
   ! variable locale
   INTEGER :: i

   ! INSTRUCTIONS
   OPEN(10, file=nomfich)
   READ(10,*) n
   CLOSE(10)

END SUBROUTINE lecture_valeur
```

```
!-----------------------------------
SUBROUTINE lecture_vecteur(nomfich,n,u)

   IMPLICIT NONE
   ! DECLARATIONS
   ! variables en arguments
   CHARACTER(len=7), INTENT(IN) :: nomfich
   INTEGER, INTENT(IN) :: n
   REAL, DIMENSION(100), INTENT(OUT) :: u
   ! variable locale
   INTEGER :: i

   ! INSTRUCTIONS
   OPEN(10, file=nomfich)
   READ(10,*) i
   DO i = 1, n
      READ(10,*) u(i)
   END DO
   CLOSE(10)

END SUBROUTINE lecture_vecteur
```

# Exemple du tri (4)

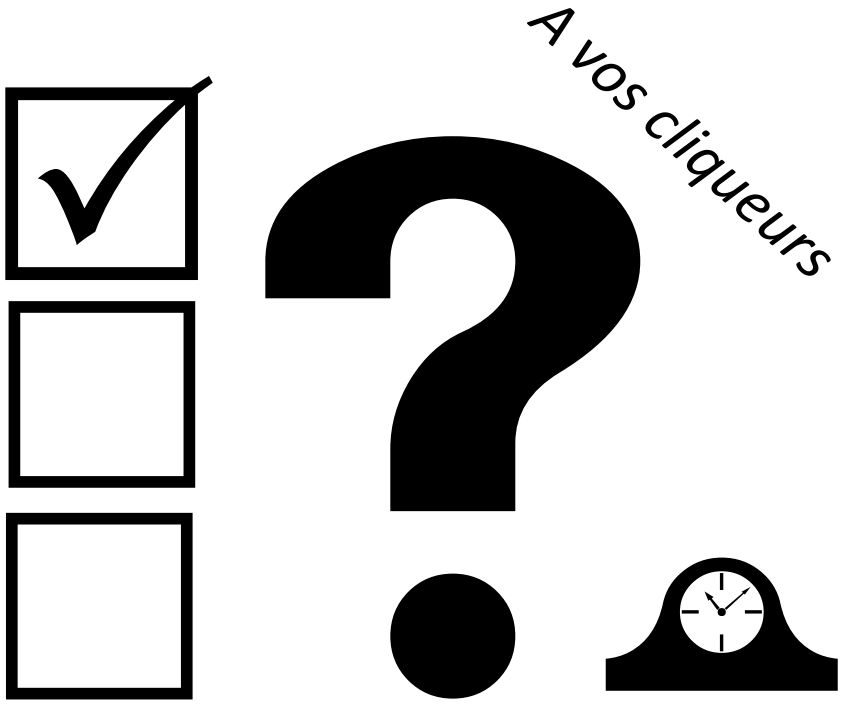
```
!-----------------------------
SUBROUTINE tri_vecteur(n,u,v)

   IMPLICIT NONE
   ! DECLARATIONS
   ! variables en arguments
   INTEGER, INTENT(IN) :: n
   REAL, DIMENSION(100), INTENT(INOUT)  :: u
   REAL, DIMENSION(100), INTENT(OUT) :: v
   ! fonctions
   REAL :: val_max
   ! variable locale
   REAL :: umax, umin
   INTEGER :: i, jmin

   ! INSTRUCTIONS
   umax = val_max(n,u)
   DO i = 1, n
        CALL recherche_min(n,u,umax,umin,jmin)
        v(i) = umin
        u(jmin) = umax
   END DO

END SUBROUTINE tri_vecteur
```

```
!--------------------
FUNCTION val_max(k,p)

   IMPLICIT NONE
   ! DECLARATIONS
   ! variables en argument
   INTEGER, INTENT(IN) :: k
   REAL, DIMENSION(100), INTENT(IN) :: p
   ! fonction
   REAL :: val_max
   ! variables locales
   INTEGER :: i

   ! INSTRUCTIONS
   val_max = p(1)
   DO i = 2, k
       IF( p(i) >= val_max ) THEN
            val_max = p(i)
        END IF
   END DO

END FUNCTION val_max
```

# Exemple du tri (4)

```
!-----------------------------------------
SUBROUTINE recherche_min(n,u,umax,umin,jmin)

   IMPLICIT NONE
   ! DECLARATIONS
   ! variables en arguments
   INTEGER, INTENT(IN) :: n
   REAL, DIMENSION(100), INTENT(IN) :: u
   REAL, INTENT(IN) :: umax
   REAL, INTENT(OUT) :: umin
   INTEGER, INTENT(OUT) :: jmin
   ! variables locales
   INTEGER :: i

   ! INSTRUCTIONS
   jmin = 1
   umin = umax
   DO i = 1, n
       IF ( u(i) <= umin ) THEN
             umin = u(i)
             jmin = i
       END IF
   END DO

END SUBROUTINE recherche_min
```

```
!------------------------------
SUBROUTINE ecriture(nomfich,n,v)

   IMPLICIT NONE
   ! DECLARATIONS
   ! variables en arguments
   CHARACTER(len=7), INTENT(IN) :: nomfich
   INTEGER, INTENT(IN) :: n
   REAL, DIMENSION(100), INTENT(IN) :: v
   ! variable locale
   INTEGER :: i

   ! INSTRUCTIONS
   OPEN(10, file=nomfich)
   WRITE(10,*) n
   DO i = 1, n
      WRITE(10,*) v(i)
   END DO
   CLOSE(10)

END SUBROUTINE ecriture
```


A vos cliqueurs

# Que mettre en arguments ?

Quiz n°1

## Quelle version est la bonne?

! Version n°1

```
SUBROUTINE toto (a, f)
  IMPLICIT NONE
  INTEGER, INTENT(IN) :: a
  INTEGER, INTENT(OUT) :: f
  INTEGER :: i
  DO i = 1, a
     f = b + c
  END DO
END SUBROUTINE toto
```

! Version n°2

```
SUBROUTINE toto (a, b, c, f)
  IMPLICIT NONE
  INTEGER, INTENT(IN) :: a, b, c
  INTEGER, INTENT(OUT) ::  f
  INTEGER :: i
  DO i = 1, a
     f = b + c
  END DO
END SUBROUTINE toto
```

! Version n°3

```
SUBROUTINE toto (a, b, c, d, e, f)
  IMPLICIT NONE
  INTEGER, INTENT(IN) :: a, b, c, d
  INTEGER, INTENT(OUT) :: f
  INTEGER :: i
  DO i = 1, a
     f = b + c
  END DO
END SUBROUTINE toto
```

# Qu'affiche ce programme ?

Quiz n°2

```
PROGRAM  tete_a_toto

   IMPLICIT NONE

   INTEGER :: f

   CALL toto ( 1, 1, 1, f)
   WRITE(*,*) 'f=',f

END PROGRAM  tete_a_toto


SUBROUTINE toto (a, b, c, f)

   IMPLICIT NONE

   INTEGER, INTENT(IN) :: a, b, c
   INTEGER, INTENT(OUT) ::  f
   INTEGER :: i

   DO i = 1, a
        f = b + c
   END DO

END SUBROUTINE toto
```

A. Rien…

B. f=1

C. f=2

D. f= (une valeur aléatoire)

E. ni A, ni B, ni C, ni D…

# Recommandations (subroutines)

- Eviter de mélanger dans le sous-programme (SP) traitement (calcul) et interactions avec l'utilisateur du programme (affichage ou saisie) : Les traitements sont plus stables que l'IHM…

- Un SP doit être une boîte noire
  ⇒ ne pas dépendre de variables globales

- Un SP ne doit pas avoir trop de paramètres
  ⇒ regrouper les paramètres dans une structure

- Un SP ne doit pas être trop long (sinon le découper)

- Un SP ne doit pas avoir trop de structures de contrôle imbriquées (sinon faire des SP dans le SP).

# Ce qu'on peut faire

- Mettre plusieurs subroutines dans une subroutine
- Mettre plusieurs fonctions dans une fonction
- Mettre plusieurs fonctions dans un subroutine

- Appeler plusieurs fois une subroutine (n'importe où)
- Appeler plusieurs fois une fonction (n'importe où)

## Suite de Fibonacci

Ecrire un programme qui détermine l'ensemble des valeurs $u_1$, $u_2$, …, $u_n$ ($n \leq 100$ étant fourni en donnée) de la suite définie comme suit: $u_1 = 1$, $u_2 = 1$, $u_n = u_{n-1} + u_{n-2}$ (pour $n>2$). Les valeurs n, $u_1$ et $u_2$ sont stockées dans le fichier formaté don_entree.dat sous la forme: « n $u_1$ $u_2$ » sur une ligne. Les résultats seront écrits dans le fichier texte don_sortie.dat sous la forme:

« Les 'n' premiers termes de la suite de Fibonacci sont:
u(1)
…
u(n) »

On utilise pour cela plusieurs sous-programmes qui, respectivement:

-> lit don_entree.dat
-> calcule les u(i)
-> écrit les résultats dans don_sortie.dat