INP TOULOUSE ENSEEIHT

Programmation impérative: Méthode de Programmation

1. Algorithmique + Fortran élémentaire

Table des matières

Définition d'un programme

❑ Définition

Un programme est une suite finie d'instructions pré-déterminées destinées à être exécutées de manière automatique par un processeur en vue d'effectuer des traitements, impliquant généralement une interaction avec son environnement.

Exemples d'environnement

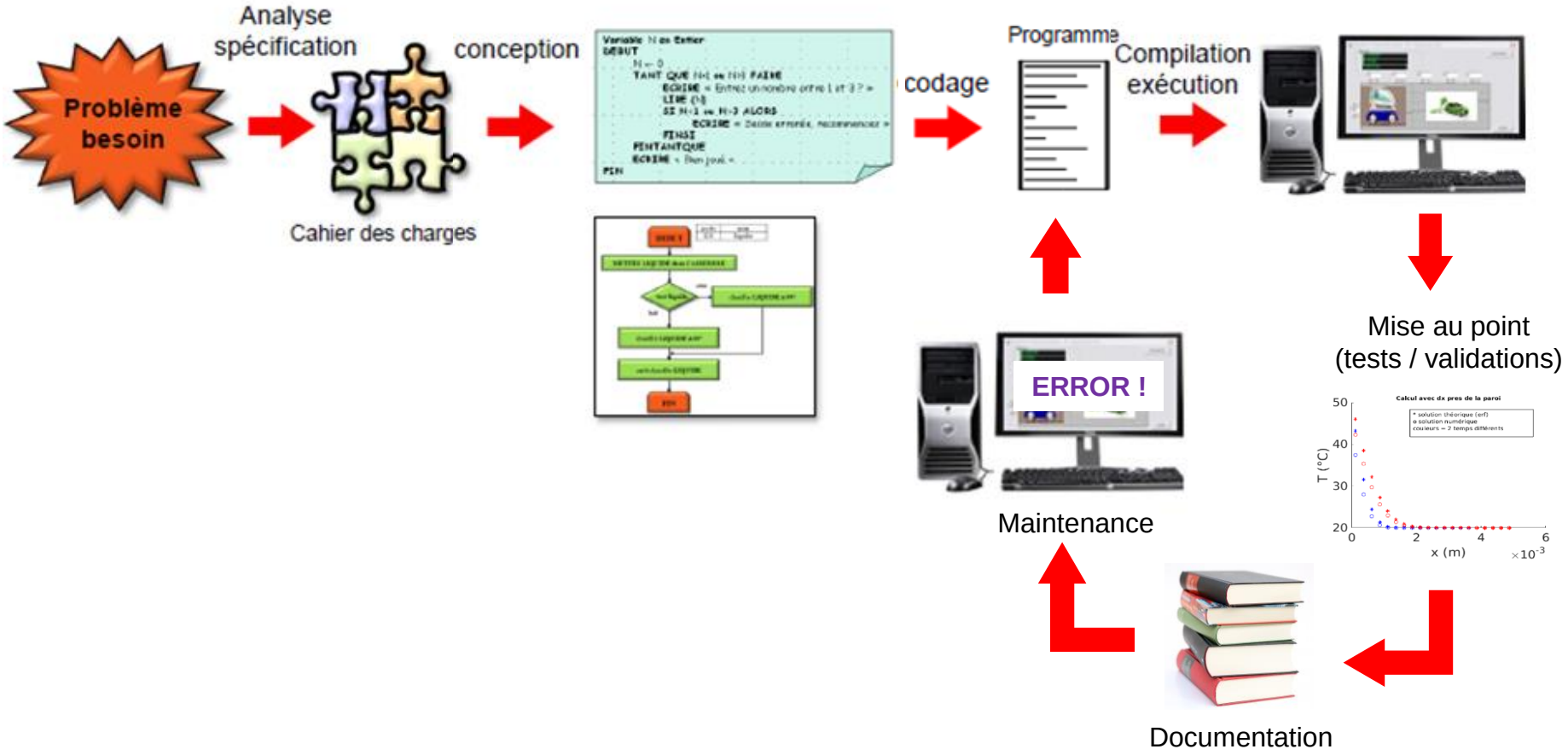
- ❑ Utilisateur humain : traitement de texte, SMS..
- ❑ Un autre système informatique : navigateur internet, guichet automatique bancaire, réseau, etc.
- ❑ Des éléments physiques : capteurs et actionneurs (ABS, régulateur vitesse).

Programme Impératif

- ❑ Programme constitué d'une suite d'ordres (actions) exécutés par un ordinateur.

Il existe d'autres types de programmes qui ne sont pas impératifs.

Formalisation d'un programme



Un programme

Objectif d'un programme :
Obtenir des

Résultats

Environnement initial :
Constitué par des

Données

Transformation :
Répond à une

Fonctionnalité

Définition d'un algorithme

Un algorithme est la description de la suite des actions qu'il faut faire réaliser à un processeur déterminé pour qu'il délivre l'ensemble des résultats à partir de l'ensemble des données

- Indépendant du langage de programmation
- Compréhensible par tous
- Contient tout ce qu'il faut pour réaliser le codage
- Description par le langage pseudo-algorithmique / organigramme

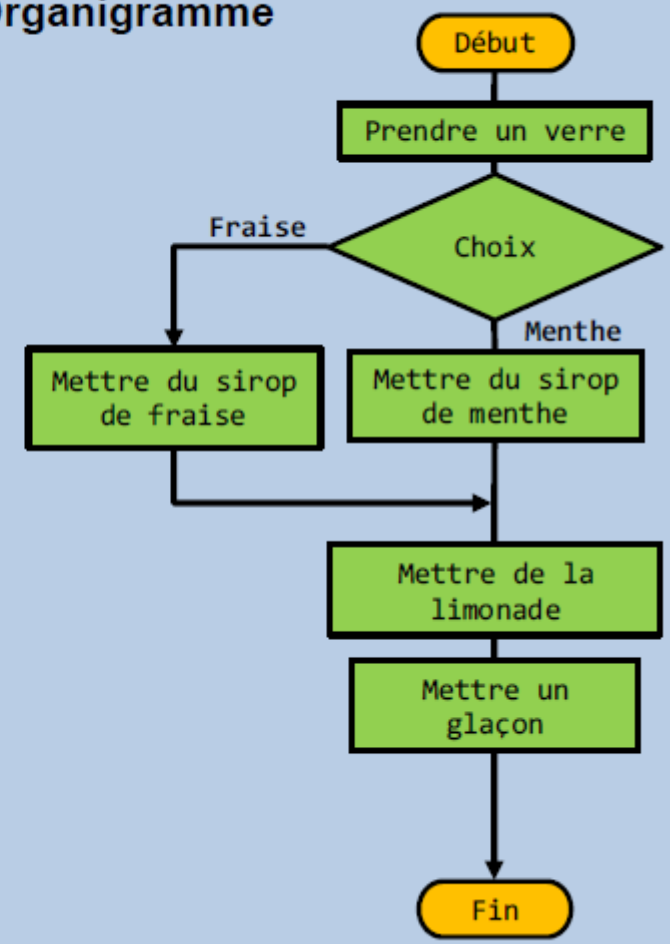
Description d'un algorithme

❑ **Exemple : Algorithme pour un diabolo (menthe ou fraise)**

Pseudo-code

```
Prendre un verre

Si CHOIX menthe

  Alors Mettre du sirop de menthe

Sinon Si CHOIX fraise

  Alors Mettre du sirop de fraise

FinSi

Mettre de la limonade
Mettre un glaçon
```

Organigramme



Principe du raffinage

- Pour aider un concepteur à concevoir un algorithme, nous proposons de suivre une démarche à étapes successives permettant d'aider à obtenir une solution en réponse à un cahier des charges.

1. Il faut exprimer l'action principale R0 que l'on veut réaliser et avec quel processeur on souhaite le faire
2. Il faut identifier les principales données manipulées. Des données supplémentaires pourront être rajoutées plus tard
3. Il faut ensuite décomposer l'action principale à l'aide d'actions plus simples notées Ri

Tant que l'on considère que les actions sont trop complexes pour être comprise par le processeur, la décomposition continue par ajout de niveaux (R1, R2, R3,…)

c'est le processus de raffinage

Ex.: préparation du diabolo menthe

1 **Processeur :** 

2 **Données :**

- Verre
- Limonade
- Sirop de menthe
- Glaçons

Version Ecrite

3

```
R0 : Préparer un diabolo menthe

R1 : Raffinage de {Préparer un diabolo menthe}
   | Prendre un verre
   | Mettre de la limonade dans le verre
   | Mettre du sirop dans le verre
   | Mettre un glaçon dans le verre

R2 : Raffinage de {Mettre de la limonade dans le verre}
   | Ouvrir le frigo
   | Prendre la limonade
   | Verser la limonade dans le verre
   | Remettre la limonade au frigo
   | Fermer le frigo
```

Trop complexe donc raffinage

Niveau 0 **Niveau 1** **Niveau 2**

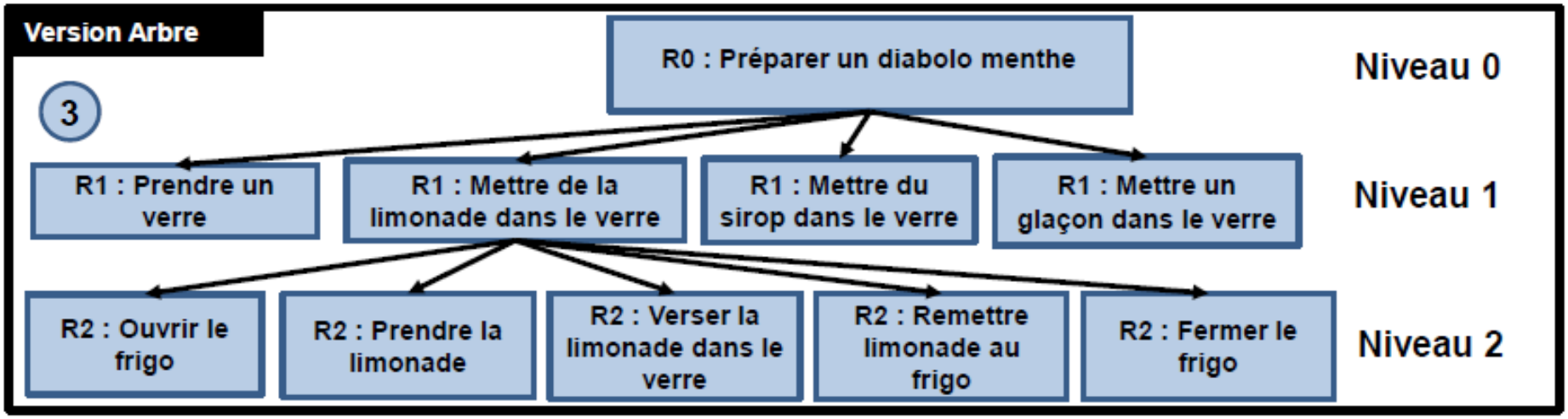
Ex.: préparation du diabolo menthe

1 **Processeur :**

Humain

2 **Données :**

- Verre
- Limonade
- Sirop de menthe
- Glaçons



❑ **Le critère d'arrêt** de la décomposition dépend du problème et de l'expérience du concepteur. Il s'agit en général d'obtenir des sous-actions correspondant à des actions élémentaires facilement réalisables par un processeur

Etape: Programmation

Quelques types de langages

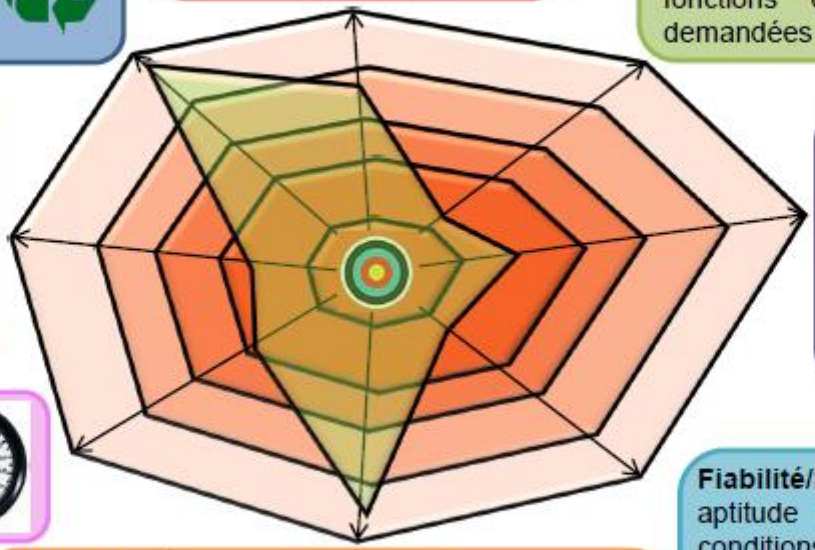
- Les langages machine : définissent le jeu d'instructions élémentaires correspondant aux capacités d'un processeur
- Les langages assembleur : gèrent les adresses logiques étiquettes, déchargent le programmeur du positionnement du programme en mémoire
- **Les langages structurés** : s'appuient sur les structures de contrôle (conditionnelle, répétition) pour éviter la profusion de branchements (et les programmes spaghettis). Structuration en sous-programmes et modules. Exemples : Fortran (1962), Pascal (1969), C (1972), Ada (1983), etc.
- Les langages objets (années 80) : regrouper les données et les traitements. Exemples : SmallTalk (1980), C++ (1983), Java (1995), Python (1990) , Fortran 90,…
- Les langages dédiés (DSL : Domain specific langage) : dédiés à des technologies spécifiques, contrairement aux langages généralistes (GSL : General purpose langage). Exemples : Matlab (1TR), VHDL, JavaScript, etc.

Critères de qualité d'un programme

Compatibilité:
facilité pour être combiné avec d'autres logiciels

Extension/Maintenance:
facilité à la maintenance, c.à.d. à une modification ou à une extension des fonctions qui lui sont demandées

Réutilisabilité:
aptitude à être réutilisé, en tout ou en partie, dans de nouvelles applications



Portabilité:
facilité pour être transféré sous différents environnements matériels et logiciels

Validité:
aptitude à remplir exactement ses fonctions, définies par le cahier des charges et les spécifications

Efficacité:
Utilisation optimales des ressources matérielles

Fiabilité/Robustesse:
aptitude à fonctionner dans des conditions anormales

Accessibilité:
facilité d'apprentissage, d'utilisation, de préparation des données, d'interprétation des erreurs et de rattrapage en cas d'erreur d'utilisation

n ! en python, en C, en Fortran

```python
def factorielle (n) :
 ########################################
# fonction factorielle
# calcule la factorielle d'un entier >= 0
# paramètre n, n entier
# précondition n >= 0
########################################
      resultat = 1
      for i in range (2, n + 1) :
               resultat = resultat * i
      return resultat



print( factorielle (3))
```

```c
#include<stdio.h>


int appel_factorielle(int n)
{
/* fonction factorielle
   calcule la factorielle d'un entier >=0
   parametre n, n entier donne
   precondition n>=0
*/
   int i,resultat=1;

   for (i=1;i<n+1;i++)  resultat=resultat*i;
   return resultat;
}


void main()
{
int fact;
fact=appel_factorielle(4);
printf("\n 4!=%d \n",fact);
};
```

```fortran
program main
  implicit none
  integer :: factorielle
  print*, factorielle(4)
end program main

function factorielle (n)
  ! calcule la factorielle de n (entier >=0)
  implicit none
  integer, intent(in) :: n
  integer :: i, factorielle
  factorielle = 1
  do i=1,n
     factorielle = factorielle * i
  end do
end function factorielle
```

Attention:
Python langage interprété
C / Fortran langage compilé

Langage interprété vs compilé

Compilateur

Programme qui transforme un code source écrit dans un langage de programmation en un autre langage informatique (langage cible)

- Passage d'un langage compréhensible par l'humain en langage machine (ie C/Fortran, Ada, …)

Interpréteur

Outil d'éxecution d'un langage informatique. Il ne produit pas de code executable, la transformation se fait à la volée et est exécutée ligne après ligne (ce qui ralenti généralement l'exécution)

- Exemple : matlab/Python

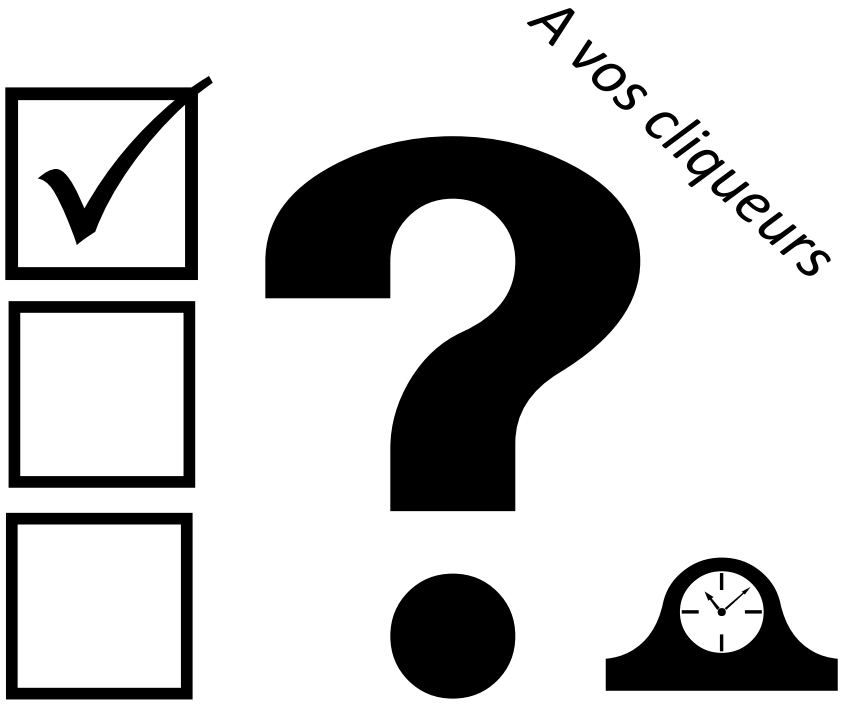
Séquentialité d'un algorithme…

Mon_algo_21

```
i ← 1
j ← 2
i ← i + j
j ← i - j
i ← i - j
```

Mon_algo_1-1

```
i ← 1
j ← 2
j ← i - j
i ← i + j
i ← i - j
```


A vos cliqueurs

Séquentialité d'un algorithme… Quiz

Mon_algo_21

```
i ← 1
j ← 2
i ← i + j
j ← i - j
i ← i - j
```

Mon_algo_1-1

```
i ← 1
j ← 2
j ← i - j
i ← i + j
i ← i - j
```

Que valent i et j ?

A. i=1 j=-1
B. i=1 j=2
C. i=2 j=1

#QDLE#Q#ABC*#60#

Séquentialité d'un algorithme… Quiz

Mon_algo_21

```
i ← 1
j ← 2
i ← i + j
j ← i - j
i ← i - j
```

Mon_algo_1-1

```
i ← 1
j ← 2
j ← i - j
i ← i + j
i ← i - j
```

Que valent i et j ?

A. i=1 j=-1
B. i=1 j=2
C. i=2 j=1

#QDLE#Q#A*BC#60#

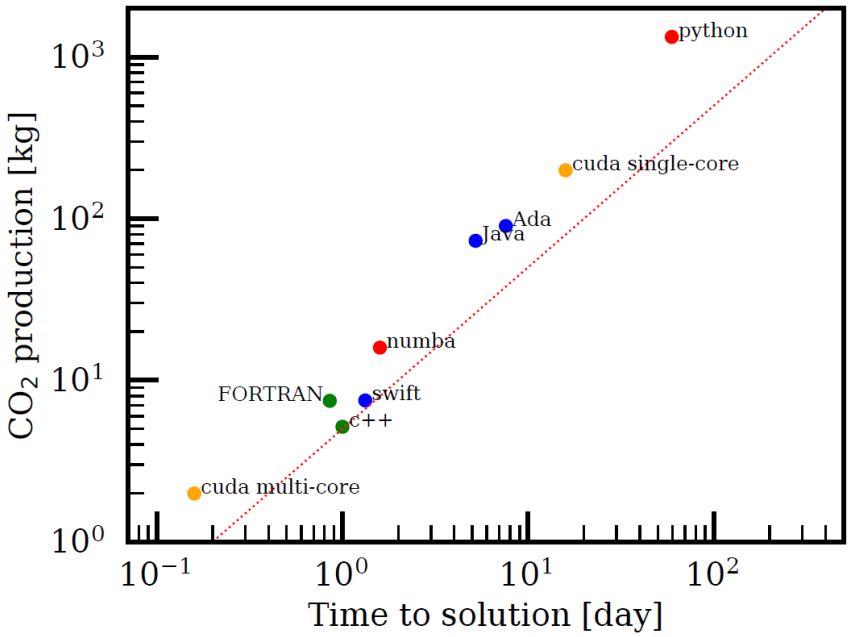
Règles de bonne programmation

- Utiliser des identificateurs significatifs et commentés pour les noms de constantes, variables, types (identificateurs de types, on en parlera plus tard)
- Toute variable doit être initialisée avant de pouvoir en utiliser la valeur.
- En principe, on donnera une valeur initiale à la variable uniquement au moment ou on en a besoin (et pas nécessairement à la déclaration)
- On choisira, pour un problème donné, les bonnes structures de contrôle (ne pas utiliser un TANT QUE à la place d'un POUR etc.)
- Indenter le code, écrire une seule instruction par ligne
- On vérifiera que les boucles de répétition sont bien écrites, à savoir :
  - La boucle se termine
  - Les variables de la boucle et de la condition ont bien une valeur
  - Le commentaire de fin de boucle sera systématiquement écrit
- Les lectures de données (ou affichages de résultats) doivent être conviviales et fiables

FORTRAN = FORmula TRANslator

Pourquoi du Fortran ?

- Langage utilisé dans 80% des logiciels de calcul scientifique (HPC)
- Langage structuré (rigueur de programmation)
- Langage « écolo »? (cf Ref [1])



Ref[1]: Zwart, The ecological impact of high-performance computing in astrophysics, Nature Astronomy, 4(9), 819-822 (2020)

FORTRAN = FORmula TRANslator

1946-48 : Instructions machines fastidieuses et peu fiables

~1950 : Assembleur

1954 : John Backus (IBM) créé le langage symbolique « Mathematical Formula Translating System », dit FORTRAN.

1956 : Premier manuel Fortran I (noms de variables, instruction, FORMAT...)

1957 : Fortran II, premiers compilateurs commerciaux (sous-programmes et fonctions compilables)

1958 : Fortran III est disponible, mais reste interne à la compagnie IBM.

1962 : Fortran IV : langage informatique des scientifiques pendant seize ans.

1966 : Fortran 66 : Création de la norme ANSI.

1978 : Fortran 77 : fin des cartes perforées début de programmation structurée

1991 : Fortran 90 : changement profond du langage: modularité, calcul vectoriel, contrôle de la précision numérique, blocs interfaces, orientation POO

1995 : Fortran 95 : caractérise fonction obsolète

2008 : Fortran 2008 : supporte programmation parrallèle

2018 : Fortran 2018 : Corrections mineures

Corps du programme

Algorithmique

/* déclarations */

Identificateur	Type	Signification

Programme principal

début

{ instructions }

fin

FORTRAN

```
PROGRAM nom_du_programme

IMPLICIT NONE

! déclarations

! instructions

END PROGRAM nom_du_programme
```

Propriétés

- Une variable doit être déclarée : nom, type et rôle (sémantique)
- On accède à une variable par son nom (identificateur)
- A la déclaration, sa valeur est indéterminée
- La valeur d'une variable est une information dynamique. Elle n'est connue qu'à l'exécution du programme.
- La valeur d'une variable est initialisée/modifiée (par instruction d'affectation)

Variables simples: les types de base…

Entier

6 bits : ± 32
16 bits / 2 octets: ± 32768
32 bits / 4 octets: ± 2×10^{9} (6 chif. signif.)
64 bits / 8 octets: ± 10^{19} (15 c.s.)

Décimal

32 bits / 4 octets: ± 10^{-38} à 10^{38} (6 c.s.)
64 bits / 8 octets: ± 2×10^{-308} à 2×10^{308} (15 c.s.)

Logique

Alphabétique (au sens large)

FORTRAN

nb d'octets

```
INTEGER :: i, j
INTEGER (KIND=8) :: k, l
! Ex.:  +4012   4012   -123

REAL :: x,y
REAL (KIND=8) :: dx,dy
! Ex. : 12.43      -0.38      -.38          4.      .27
!        12.43E0   1.243E1   1.243e+1

LOGICAL :: etu
! Ex.:   .FALSE.    .TRUE.
```

nb de caractères

```
CHARACTER (len=28) :: nom
! Ex.: 'toto 1'      'je sais qu on ne sait jamais'
```

Ecriture / Lecture à l'écran

Algorithmique

Ecriture

Ecrire(a)
Ecrire(a,i)
Ecrire(« Le resultat est: »,a)

Lecture

Lire(a)
Lire(a,i)

FORTRAN

```
INTEGER :: a, i
CHARACTER (len=20) :: phrase
```

```
WRITE(*,*) a
WRITE(*,*) a, i
WRITE(*,*) ″Le resultat est: ″ , a
```

⟺ PRINT*, a

raccourci d'écriture

```
READ(*,*) a
READ(*,*) a, i
```

⟺ READ*, a

raccourci d'écriture

Exemple de la facture

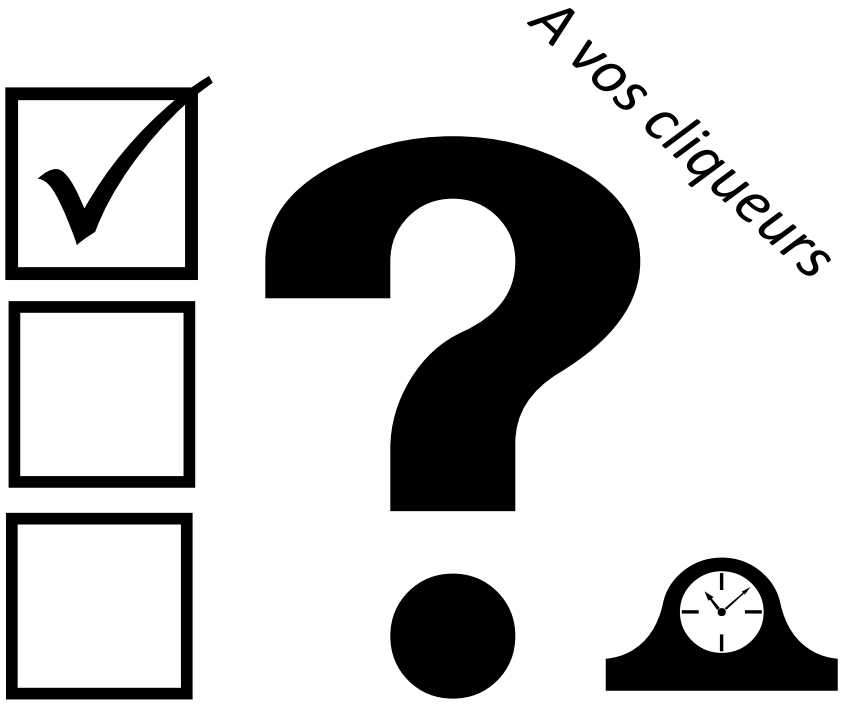
Algorithmique

Facture

Lire (p1)
Lire (n1)
pht1 ← n1 * p1
Lire (p2)
Lire (n2)
pht2 ← n2 * p2
pht ← pht1 + pht2
pttc ← pht * 1.186
Ecrire (pttc)

FORTRAN

```
PROGRAM FACTURE
   IMPLICIT NONE

   INTEGER :: n1, n2
   REAL :: p1, p2, pht1, pht2, pht, pttc

   WRITE(*,*) "Prix HT unitaire du 1er type d'articles ?"
   READ(*,*) p1
   WRITE(*,*) "Nombre d'articles du 1er type ?"
   READ(*,*) n1
   pht1 = real(n1) * p1
   WRITE(*,*) "Prix HT unitaire du 2eme type d'articles ?"
   READ(*,*) p2
   WRITE(*,*) "Nombre d'articles du 2eme type ?"
   READ(*,*) n2
   pht2 = real(n2) * p2
   pht = pht1 + pht2
   pttc = pht * 1.186
   WRITE(*,*) "La facture est de: ", pttc, " euros"

END PROGRAM FACTURE
```


A vos cliqueurs

Quel programme est correct?

Quiz

A.

```
program TOTO

implicit none

integer :: a

write(*,*) "Donner la valeur de a (entier)"
read(*,*) a

print*, "La valeur de a est", a

end program TOTO
```

B.

```
program TATA

implicit none

integer :: a

print*, 'Donner la valeur de a (entier)'
read*, a
write*, "La valeur de a est", a

end program TATA
```

#QDLE#Q#A*B#60#

Opérateurs arithmétiques 1/2

Algorithmique	FORTRAN
addition, soustrac.°, multipl.°, division	+ - * /
	! Ex.: 7/5 donne la valeur 1
	! 7./5. donne la valeur 1.4
	! 7./5 donne une valeur fonction du compilateur…
Attention: Faire des opérations entre objets de même type uniquement !	 à éviter
moins (unaire)	- ! Ex.: -3 -1.2 -98.e-2

Opérateurs arithmétiques 2/2

Algorithmique

élévation à la puissance

**

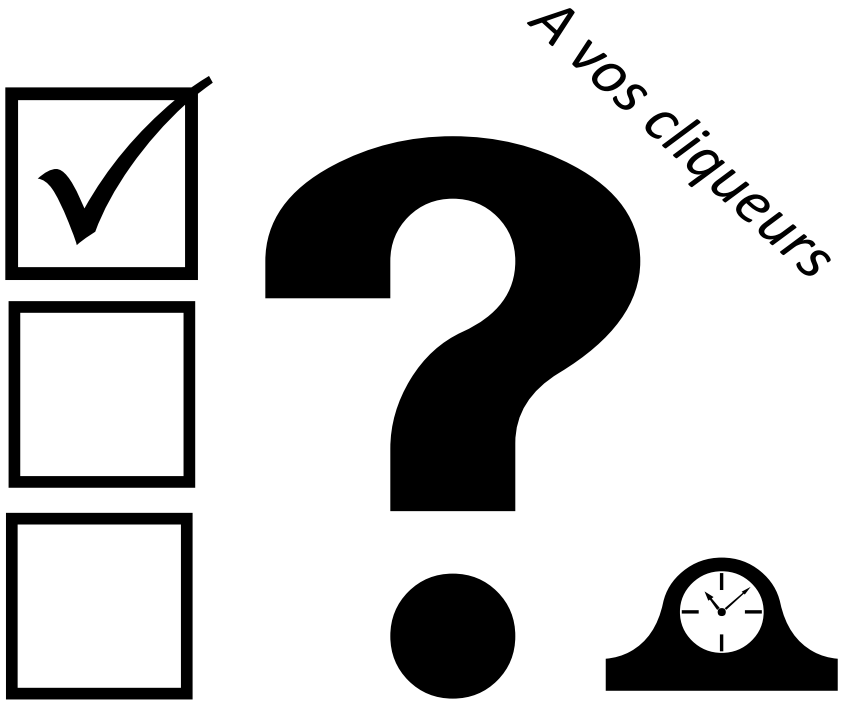
```
! Ex.:    2**3        donne 8 (entier)
!         2**(-3)     donne 0 (entier)
!         2.**3
!         2**3.       } donnent 8. (réel)
!         2.**3.
!         2.**(-3)
!         2**(-3.)    } donnent 0.125 (réel)
!         2.**(-3.)
!         -2**3       donne -8 (entier)
!         (-2.)**3    donne -8. (réel)
!         (-2.)**3.
```

Remarque: $e^{3\ \ln(-2.)}$ n'est pas possible

Affectation

Algorithmique	FORTRAN
$a \leftarrow b$	a = b
$i \leftarrow i+1$	i = i+1
En programmation a = a + b est tout-à-fait possible même avec $a \neq 0$ et $b \neq 0$ …	a = a + b

Attention: Faire des affectations entre objets de même type uniquement !


A vos cliqueurs

Qu'affiche ce programme ?

Quiz

```
PROGRAM TOTO
IMPLICIT NONE
INTEGER :: n, p, q
REAL :: x

  n = 3
  p = 5
  x = 1.2
  PRINT*, x + n/p
  q = p + x*n
  PRINT*, p + x*n
  PRINT*, q

END PROGRAM TOTO
```

	A.	B.	C.	D.
1ère ligne	1.8	1.8	1.2	1.2
2ème ligne	8.6	8	8.6	8.6
3ème ligne	8	8	8	8.6

#QDLE#Q#ABC*D#60#

Problème de transtypage

```
PROGRAM TOTO
IMPLICIT NONE
INTEGER :: n, p, q
REAL :: x

  n = 3
  p = 5
  x = 1.2
  PRINT*, x + n/p
  q = p + x*n
  PRINT*, p + x*n
  PRINT*, q

END PROGRAM TOTO
```

réel + entier/entier !!!

entier = entier + réel*entier !!!

entier + réel*entier !!!

C.

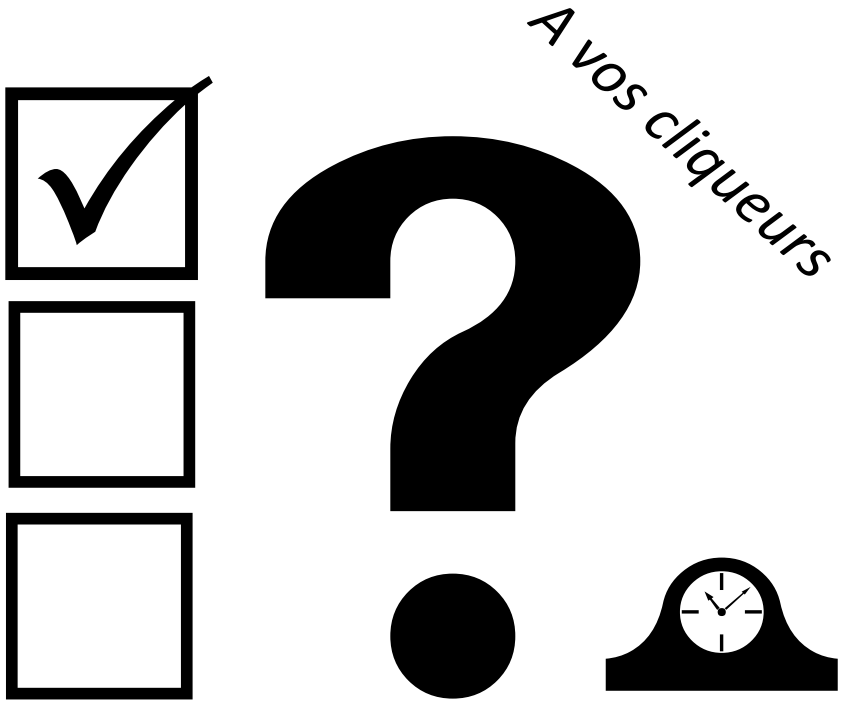
1.2

8.6

8

Solution au problème de transtypage

```
PROGRAM TOTO
IMPLICIT NONE
INTEGER :: n, p, q
REAL :: x

  n = 3
  p = 5
  x = 1.2
  PRINT*, x + real(n)/real(p)
  q = p + int(x)*n
  PRINT*, p + int(x)*n
  PRINT*, q

END PROGRAM TOTO
```


A vos cliqueurs

Solution au problème de transtypage

```
PROGRAM TOTO
IMPLICIT NONE
INTEGER :: n, p, q
REAL :: x

  n = 3
  p = 5
  x = 1.2
  PRINT*, x + real(n)/real(p)
  q = p + int(x)*n
  PRINT*, p + int(x)*n
  PRINT*, q

END PROGRAM TOTO
```

Quiz

	A.	B.	C.	D.
	1.8	1.8	1.2	1.2
	8.6	8	8.6	8.6
	8	8	8	8.6

#QDLE#Q#AB*CD#60#

Expressions logiques

Algorithmique	FORTRAN
Comparaison d'entiers/réels	== ! égal à /= ! différent de < <= > >=
Comparaison de logiques	.EQV. .NEQV.

Attention #1: Comparaisons entre objets de même type uniquement !

```
! Ex.:   n == m    (n, m entiers)
! Ex.:   x == y    (x, y réels)
```

Attention #2: tests d'égalité entre réels à éviter absolument !

! 2 réels ne sont égaux qu'aux erreurs de troncature près…

Connecteurs logiques: et / ou — .AND. .OR.

Traitement conditionnel 1/5

Algorithmique

Si (condition)
Alors
|

FORTRAN

```
IF (expression_logique) THEN

   instruction 1
   instruction 2
   …
   instruction n

END IF
```

Traitement conditionnel 2/5

Algorithmique

si (condition)

Alors

|

Sinon

|

FORTRAN

```
IF (expression_logique) THEN

   instruction 1
   instruction 2
   …
   instruction n

ELSE

   instruction 1
   instruction 2
   …
   instruction n

END IF
```

Traitement conditionnel 3/5

Algorithmique

Si (condition 1)
  Alors
    |

Sinon si (condition 2)
  Alors
    |

Sinon
    |

FORTRAN

```
IF (expression_logique) THEN
  instruction 1
  …
  instruction n
ELSE IF (expression_logique) THEN
  instruction 1
  …
  instruction n
ELSE
  instruction 1
  …
  instruction n
END IF
```

Traitement conditionnel 4/5

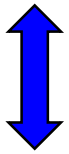
Algorithmique

Version classique

Raccourci d'écriture
(quand instruction simple uniquement)

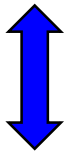
FORTRAN

```
IF (expression_logique) THEN
   instruction simple
END IF
```



```
IF (expression_logique) instruction_simple
```

```
IF (a==0) THEN
   WRITE(*,*) "a est egal a 0"
END IF
```



```
IF (a==0) WRITE(*,*) "a est egal a 0"
```

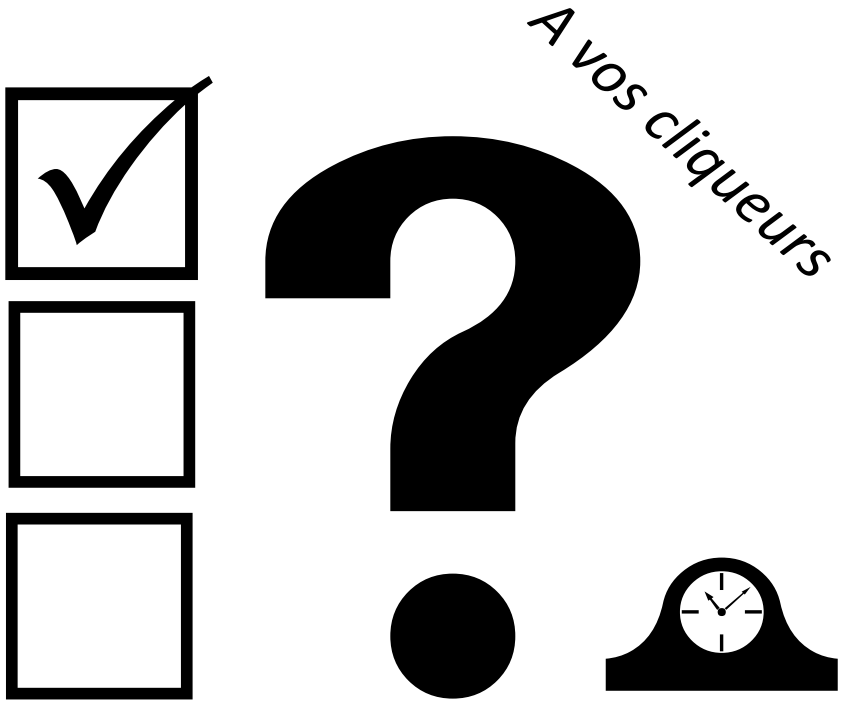
Traitement conditionnel 5/5

Algorithmique

Variante
utilisant la commande
SELECT CASE

FORTRAN

```
SELECT CASE (n)
  CASE (0)                    ! n = 0
    print *, 'eliminatoire'
  CASE (5,6,7,8,9,10)         ! n = 5 à 10
    print *, 'mention passable'
  CASE (11:15)                ! n = 11 à 15
    print *, 'mention bien'
  CASE (16:20)                ! n = 16 à 20
    print *, 'mention tres bien'
  CASE DEFAULT                ! autres cas
    print *, 'pas de mention'
END SELECT
```


A vos cliqueurs

Qu'affiche ce programme à la fin si a='oui' b='non' c='oui' age=16 ?

Quiz n°1

INP TOULOUSE ENSEEIHT

```
PROGRAM RESTO
IMPLICIT NONE

 CHARACTER (LEN=3) :: a, b, c
 INTEGER :: age

PRINT*, 'Voulez boire (oui/non)?'
READ*, a

IF (a == 'oui') THEN
   PRINT*, 'Une boisson alcolisee (oui/non)?
   READ*, b
   IF (b == 'oui') THEN
        PRINT*, 'Donner votre age'
        READ*, age
    END IF
END IF

PRINT*, 'Voulez-vous manger (oui/non)?'
READ*, c

IF (b == 'oui' .AND. age < 18) THEN
    PRINT*, 'Boire avant 18 ans est interdit'
ELSE IF (a == 'oui' .AND. c == 'oui') THEN
    PRINT*, 'Nous avons des menus'
ELSE IF (a == 'oui' .OR. c == 'oui') THEN
    PRINT*, 'Bon appetit'
ELSE
    PRINT*, 'Au revoir'
END IF

END PROGRAM RESTO
```

A. 'Boire avant 18 ans est interdit'

B. 'Nous avons des menus'

C. 'Bon appetit'

D. 'Au revoir'

E. 'Nous avons des menus'
'Bon appetit'

Qu'affiche ce programme si l'utilisateur entre 1. et 1. ?

Quiz n°2

```
program egalite

implicit none

real :: x,y

print*,"Donner deux valeurs réelles"
read*, x,y

if (x==y) then
    print*,"Les deux valeurs sont égales"
else
    print*,"Les deux valeurs sont différentes"
end if

end program egalite
```

A. 'Les deux valeurs sont différentes'

B. 'Les deux valeurs sont égales'

C. On ne sait pas

#QDLE#Q#ABC*#30#

Solution au problème de test d'égalité sur des réels

```
program inegalite

implicit none

real :: x,y

print*,"Donner deux valeurs réelles"
read*, x,y

if (abs(x-y)<=1.e-8) then
    print*,"Les deux valeurs sont égales à 10^-8 près"
else
    print*,"Les deux valeurs sont différentes"
end if

end program inegalite
```

Traitement itératif 1/3

Algorithmique

Si nombre d'itérations CONNU

Pour i de 1 à n faire

    |
    |
    |

FORTRAN

```
INTEGER :: i, n

DO i = 1, n

   instruction 1
   …
   instruction k

END DO
```

Traitement itératif 2/3

Algorithmique

Variante
(boucle à progression décroissante)

Pour i **de** n **à** 1 **faire**

à utiliser avec précautions…

FORTRAN

```
INTEGER :: i, n
DO i=n,1,-1
   instruction 1
   …
   instruction k
END DO
```

Cas général

```
INTEGER :: i, nd, na, pas
DO i=nd,na,pas
   instruction 1
   …
   instruction k
END DO
```

Traitement itératif 3/3

Algorithmique

Si nombre d'itérations INCONNU

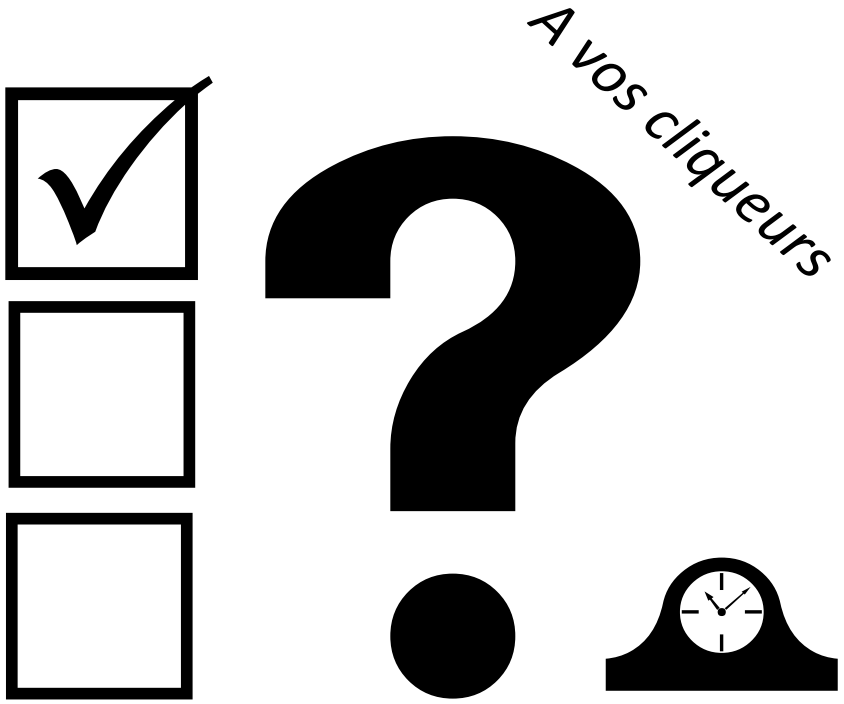
Tant que (condition) **faire**

répété tant que la condition est « vraie »

FORTRAN

```
DO WHILE (expression_logique)

   instruction 1
   …
   instruction n

END DO
```

Attention aux boucles infinies…


A vos cliqueurs

Qu'affiche ce programme ?

Quiz n°1

```
program boucle

implicit none

integer :: i

do i=1,3
   print*,"Coucou"
end do

end program boucle
```

A.	B.	C.	D.
Coucou Coucou	Coucou Coucou Coucou	Coucou Coucou Coucou Coucou	Ni A, ni B, ni C

#QDLE#Q#AB*CD#60#

Qu'affiche ce programme ?

Quiz n°2

```
program boucle_bis

implicit none

integer :: i

i=0
do while (i<=3)
   print*,"Coucou"
   i=i+1
end do

end program boucle_bis
```

A.	B.	C.	D.
Coucou	Coucou	Coucou	Ni A, ni B, ni C
Coucou	Coucou	Coucou	
	Coucou	Coucou	
		Coucou	

#QDLE#Q#ABC*D#60#

Exercice

Suite de Fibonacci

Ecrire un programme en FORTRAN qui détermine la énième valeur u_n (n 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)

IMPLICIT NONE, pour quoi faire ?

Si une variable commençant par i, j, k, l, m, n est non-déclarée
→ FORTRAN affecte implicitement le type INTEGER
Pour les autres variables non-déclarées
→ FORTRAN affecte implicitement le type REAL

Exemple sans IMPLICIT NONE

```
PROGRAM TOTO
  INTEGER :: k, nbre
  nbre = 5
  k = nbr + 1
  PRINT*, k
END PROGRAM TOTO
```

- Pas d'erreur à la compilation
- 1er lancement: « 1105920329 »
- 2nd lancement: « -459992759 »
- 3ème lancement: « -1909489335 »

Exemple avec IMPLICIT NONE

```
PROGRAM TOTO
  IMPLICIT NONE
  INTEGER :: k, nbre
  nbre = 5
  k = nbr + 1
  PRINT*, k
END PROGRAM TOTO
```

- Message d'erreur à la compilation

MAJUSCULES et minuscules en Fortran…

Le fortran ne fait PAS DE DISTINCTION entre majuscules et minuscules !!!

Toto	=	**TOTo**	=	**TotO**	=	**toTo**	=	
TOTO	=	**TOtO**	=	**tOTo**	=	**totO**	=	
Toto	=	**ToTO**	=	**tOtO**	=	**Toto**	=	
tOto	=	**tOTO**	=	**toTO**	=	**ToTo**		

Mise en œuvre d'un algorithme

Comment UTILISER un programme lorsqu'il est écrit?

- Il faut l'EXECUTER en créant un EXECUTABLE obtenu en COMPILANT le code source avec un COMPILATEUR (libre ou commercial)
- Le code source doit se trouver dans un fichier dont l'extension est « .f90 »

Qu'est-ce la compilation?

1° Etape de création de fichiers objets correspondant aux programmes. Le code source est analysé, et s'il est correct, un ensemble d'instructions processeur correspondant aux instructions du code source est généré dans un fichier objet (d'extension *.o).

2° Etape d'édition de liens liant les instructions desfichiers objets aux librairies correspondantes afin de créer le fichier exécutable (car on utilise des fonctions qui sont préprogrammées: lire/écrire un caractère, fonctions mathématiques, etc, et qui sont regroupées dans des librairies).

Compilation (ligne de commandes)

FORTRAN

```
! Code source dans le fichier « prog.f90 »
PROGRAM COUCOU
  IMPLICIT NONE
  PRINT*, 'Coucou'
END PROGRAM COUCOU

! Compilation (version très courte)
> ls
> prog.f90
> gfortran prog.f90
> ls
> prog.f90   a.out
```

FORTRAN

```
! Compilation (version courte)
> ls
> prog.f90
> gfortran prog.f90 –o coucou.exe
> ls
> prog.f90   coucou.exe

! Compilation (version longue)
> ls
> prog.f90
> gfortran prog.f90 –c
> gfortran prog.o –o coucou.exe
> ls
> prog.f90   prog.o   coucou.exe
```

Mise en œuvre d'un algorithme

Programmation

```
! Code source dans le fichier « prog.f90 »
PROGRAM COUCOU
  IMPLICIT NONE
  PRINT*, 'Coucou'
END PROGRAM COUCOU
```

Compilation (lignes de commandes)

```
! Compilation (ligne de commandes)
> ls
> prog.f90
> gfortran prog.f90 –o coucou.exe
> ls
> prog.f90   coucou.exe
```

Exécution

```
! Exécution
> ./coucou.exe
> Coucou
```