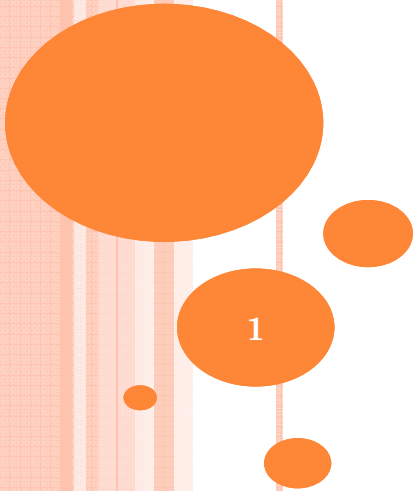


Cours de Programmation

Stéphanie Combettes

PRÉSENTATION DU COURS DE PROG

- **Enseignement de prog. divisé en**
  - Cours - TD : début de semaine
  - TP (sur machine) : fin de semaine
  - Tests hebdomadaires à faire

- **Equipe enseignante**
  - C-TD : Stéphanie Combettes, Samir Medjiah, Philippe Latu et Régis Nohé
  - TP : Les mêmes + Claude Cousturian, Isabelle Silvain et Luca Sartori

PRÉSENTATION DU COURS DE PROG -

- **Tests sous « Moodle »**
  - Aide au travail personnel
  - 1 test à faire par semaine ouvert pour 15 jours

     Ne pas attendre le dernier jour !!!
- **Examens – notes :**
  - 1 partiel écrit
  - 1 examen de TP
  - 3 examens de TD
  - 1 note de suivi de TP
  - 1 note des tests Moodle

  ➢ 3 Notes finales :
  - 1 note d'Examen écrit
  - 1 note de TD
  - 1 note de TP
- **Projet de fin de semestre** : 1 semaine pour tout changer en prog
- **Planning du semestre** disponible sous Moodle

Chapitre 1
Introduction

- Qu'est-ce que la programmation ? Un programme ?
- Quels sont les objectifs de ce cours ?
- Qu'est-ce qu'un algorithme ?
- Qu'est-ce qu'une chaîne de développement ?

QU'EST-CE QUE LA PROGRAMMATION ?

- La programmation : établissement d'un programme

- Programme : suite d'**actions** permettant d'atteindre un **objectif**
  - Les actions
    - Réalisées en **séquence** (i.e. dans l'ordre de leur écriture dans le programme)
    - Actions **élémentaires** ou **complexes** (ou structurées)
  - Des **conditions** évaluées pour déterminer la suite des actions à exécuter
  - Prise en compte des diverses situations (voire toutes) pour que l'objectif soit toujours atteint
  - Limité par un début et une fin

- Différence entre la **description** et la **réalisation** d'un programme
  - La description prend en compte TOUTES les situations possibles tandis que l'exécution fournit UNE réalisation dans un cas particulier

Exemple de programme

- Passer un bon après-midi

après_déjeuner

aller en ville

visiter un musée

SI (il pleut), aller au cinéma

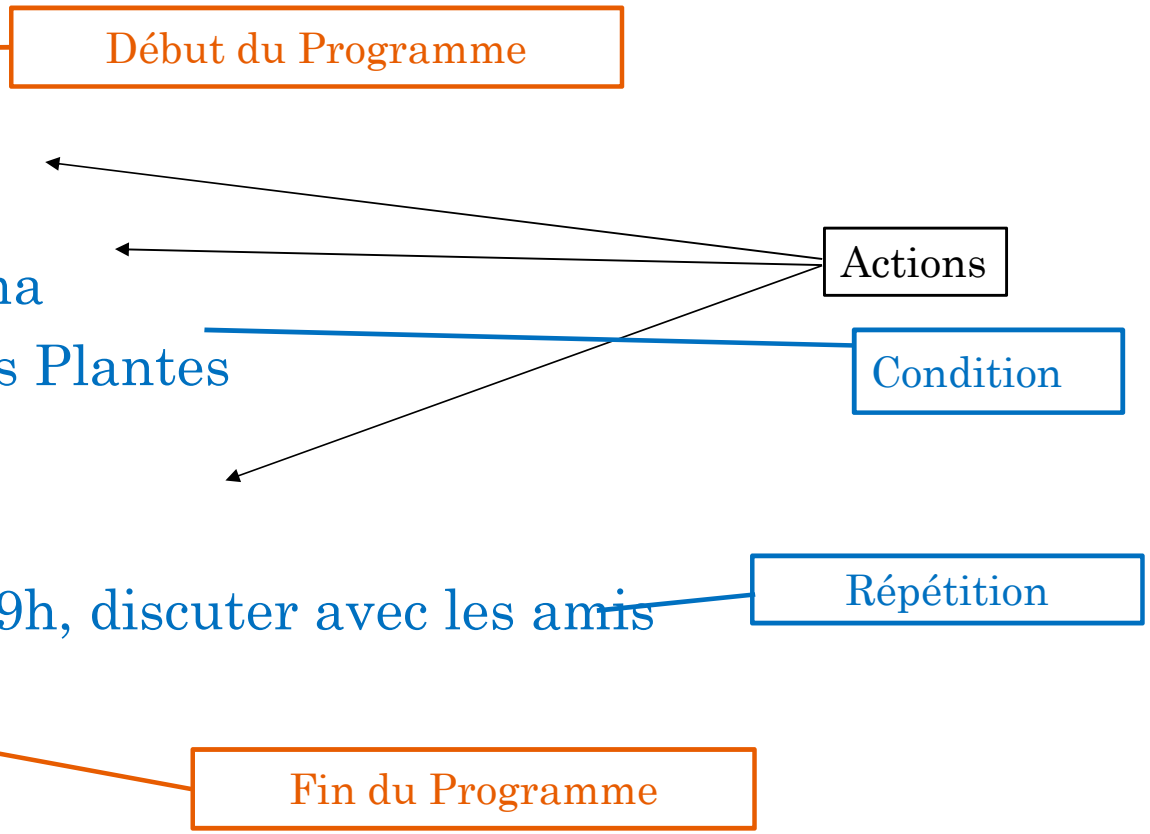
SINON aller au Jardin des Plantes

retrouver ses ami.e.s

boire un café

TANT QUE ce n'est pas 19h, discuter avec les amis

retour_maison



Objectif du cours de programmation

- Ecrire un programme en notation algorithmique (ou sous forme d'organigramme)
- Traduire cet algorithme en langage de programmation

- C'est-à-dire, pour le programmeur,
  - Savoir expliciter son raisonnement
  - Savoir formaliser son raisonnement
  - Concevoir et écrire des **algorithmes**
  - Traduire l'algorithme en langage de programmation

ALGORITHME : QUELQUES DÉFINITIONS

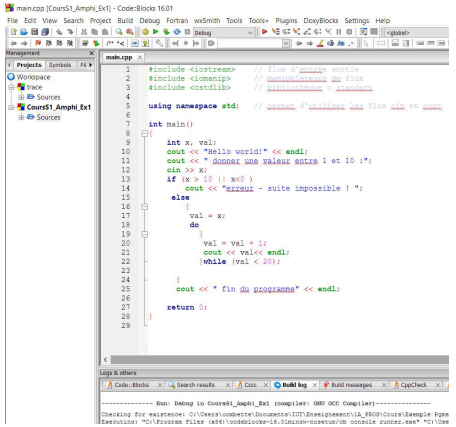
- **Un algorithme :** séquence d'instructions qui décrit comment résoudre un problème particulier

- Traduction de l'algorithme dans un langage de programmation → compréhensible par un compilateur → notion de chaîne de développement

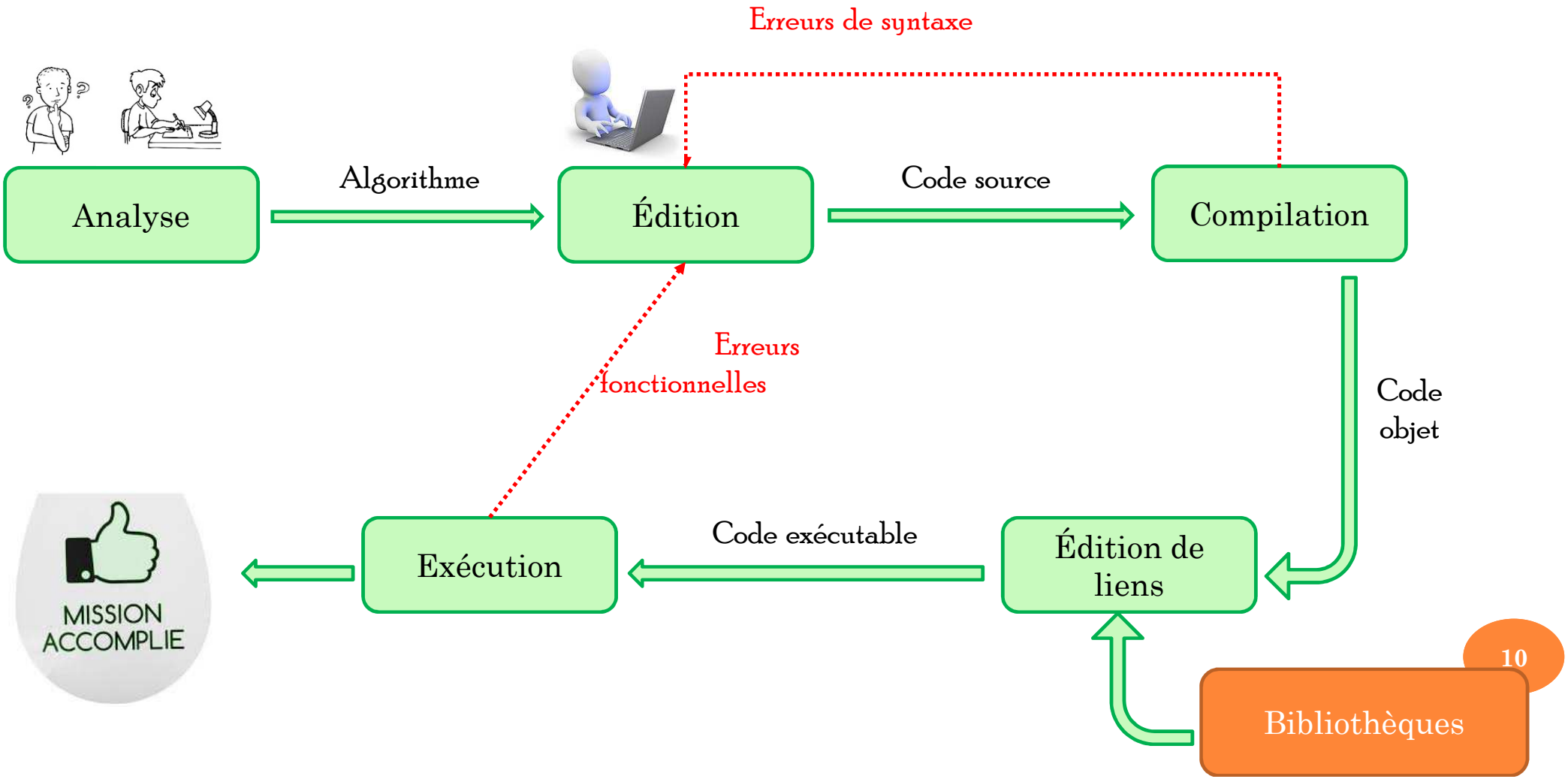
- L'algorithme permet d'établir un programme qui peut ensuite être exécuté pour effectuer le traitement souhaité
- Un algorithme ne dépend pas
  - du langage dans lequel il sera traduit
  - de la machine qui exécutera le programme

EXEMPLE :

```
DEBUT_PROGRAMME
    |
    | DECLARATIONS
    |     | Entier : x, val
    |     |
    | ACTIONS
    |     |
    |     | ECRIRE_ECRAN  " Hello, faisons une incrémentation ", FDL
    |     | ECRIRE_ECRAN  " Donner une valeur entre 1 et 10 : "
    |     | LIRE_CLAVIER x
    |     | SI ( x>10 ou x < 1) ALORS
    |     |     | ECRIRE_ECRAN « erreur – suite impossible »
    |     |     |
    |     | SINON
    |     |     | val ← x
    |     |     | REPETER
    |     |     |       | val ← val + 1     % incrémentation de val
    |     |     |       | ECRIRE_ECRAN val, FDL
    |     |     |TANT_QUE (val <20)
    |     | FSI
FIN_PROGRAMME
```



Chaîne de développement d'un programme



Chapitre 2
Ecrire un algorithme (ou un programme)

1. Données et Structure d'un algo
2. Opérateurs
3. Actions simples
4. Actions structurées

Données : l'information manipulée

- Un programme manipule des informations
- Les actions peuvent modifier et/ou dépendre d'**informations** ou de **données**

- Données
  - constantes ou variables
  - identifiées par un nom (une étiquette)
  - typées

DONNÉES : L'INFORMATION MANIPULÉE 2/4

- **Variable** : donnée dont la valeur peut varier au cours de l'exécution
- **Constante** : donnée dont la valeur ne peut pas varier durant l'exécution
- **Identificateur** d'une variable ou d'une constante
  - Nom **UNIQUE** dans le programme
  - Suite de lettres, chiffres, _ , $ de longueur quelconque
  - Différences entre majuscules et minuscules
  - 1er symbole du nom : une lettre obligatoirement
  - Pas de caractères accentués
- Convention d'écriture
  - Constante : identificateur écrit entièrement en MAJUSCULES
  - Variable : identificateur commence par une minuscule

DONNÉES : L'INFORMATION MANIPULÉE 3/4

- **Type** d'une donnée : ensemble des valeurs possibles
  - Entier signé : compris entre -32768 et 32767 (exemple de codage sur 2 octets)
  - Réel signé : compris entre $-3,4*10^{38}$ et $3,4*10^{38}$
  - Caractère : table ASCII → correspondance entre combinaison binaire et caractère
  - Booléen : vrai ou faux
- On distingue
  - les types simples : entier, réel, caractère et booléen
  - les types composés : tableaux, enregistrements, listes
- Déclaration d'une variable

type : identificateur

Exemples :

Entier : niveauEtage ⇒ pourra avoir comme valeurs : -2 ou 58 mais pas 33,5

Caractere : carac ⇒ pourra avoir comme valeurs 'A', 'g', '?' ou '5' mais pas 5

DONNÉES : L'INFORMATION MANIPULÉE 4/4

- Structure d'un algorithme

DECLARATIONS

NOMS ET TYPES DES DIFFÉRENTES CONSTANTES OU VARIABLES UTILISÉES (+ D'AUTRES INFORMATIONS)

ACTIONS

SUITE DES ACTIONS À RÉALISER POUR ATTEINDRE L'OBJECTIF

FIN ACTIONS

Les indentations (**traits verticaux et décalages**) sont **importantes**

EXEMPLE :

```
DEBUT_PROGRAMME
    |
    | DECLARATIONS
    |       | Entier : x, val
    |       |
    | ACTIONS
    |       |
    |       | ECRIRE_ECRAN  " Hello, faisons une incrémentation ", FDL
    |       | ECRIRE_ECRAN  " Donner une valeur entre 1 et 10 : "
    |       | LIRE_CLAVIER x
    |       | SI ( x>10 ou x < 1) ALORS
    |       |       | ECRIRE_ECRAN « erreur – suite impossible »
    |       |       |
    |       | SINON
    |       |       | val ← x
    |       |       | REPETER
    |       |       |       | val ← val + 1     % incrémentation de val
    |       |       |       | ECRIRE_ECRAN val, FDL
    |       |       |TANT_QUE (val <20)
    |       | FSI
FIN_PROGRAMME
```

Opérateurs et Expressions

- Servent à combiner/modifier les valeurs de données
- Dépendent du type des données
- Opérateurs de base :
  - +, -, *, /, et, ou, <, >, != (différent), == (égal)
- Une **expression** est une combinaison d'opérateurs et de variables, ou une variable seule
  - Exemples :

(2*a + b)*33 valeur

ACTIONS SIMPLES

Affectation

- Attribuer (ou affecter ou assigner) la valeur d'une expression à une variable
- Symbole de l'affectation : ←
- Exemple :

```
nombre ← 0
nombre ← j+3
nombre ← nombre +2
```

(nombre "reçoit" 0)
(nombre "reçoit" j+3)
(nombre "reçoit" nombre+2)

⇒ *Accumulation*

Le ***sens de l'affectation*** est important lors de la notation

Le **type de l'expression *et* le type de la variable** recevant le résultat doivent correspondre

ACTIONS SIMPLES 2/3

- **Lecture**
  - Lire une valeur donnée par l'utilisateur (via le clavier) au programme et l'affecter à une variable

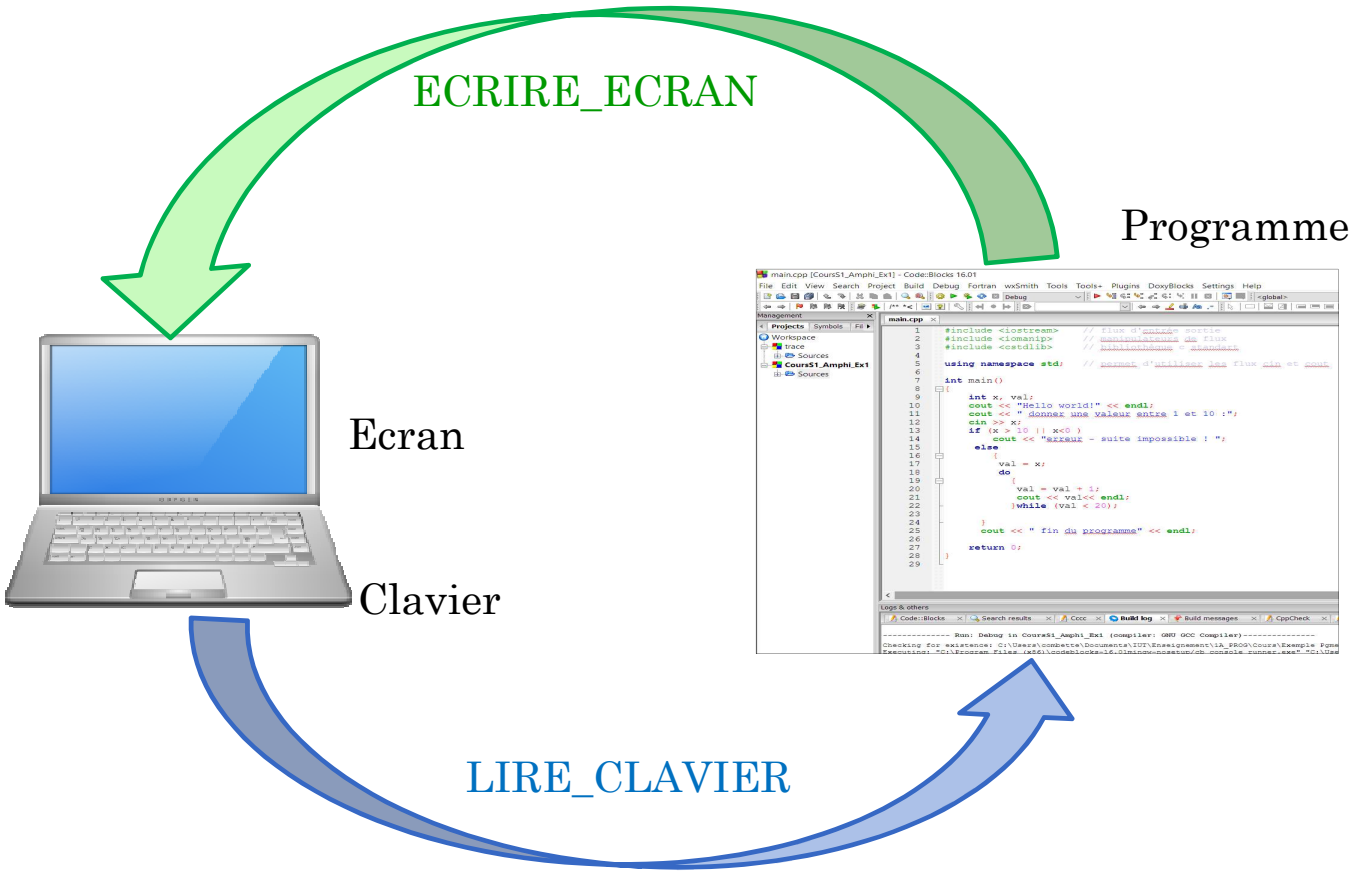
    Lire_Clavier variable1

- **Ecriture**
  - Afficher la valeur d'une donnée (ou un message) du programme à l'utilisateur (via l'écran)

    Ecrire_Ecran variable1 : affiche à l'écran la valeur de *variable1*

    Ecrire-Ecran ***"Bonjour"*** : affiche à l'écran le message Bonjour

LES ACTIONS SIMPLES 3/3



Programmeur :
Se placer du **point de vue du programme**

Programmeur ≠ Utilisateur

Jonglage entre les 2 : Programmeur lors du codage

Utilisateur lors des tests du programme

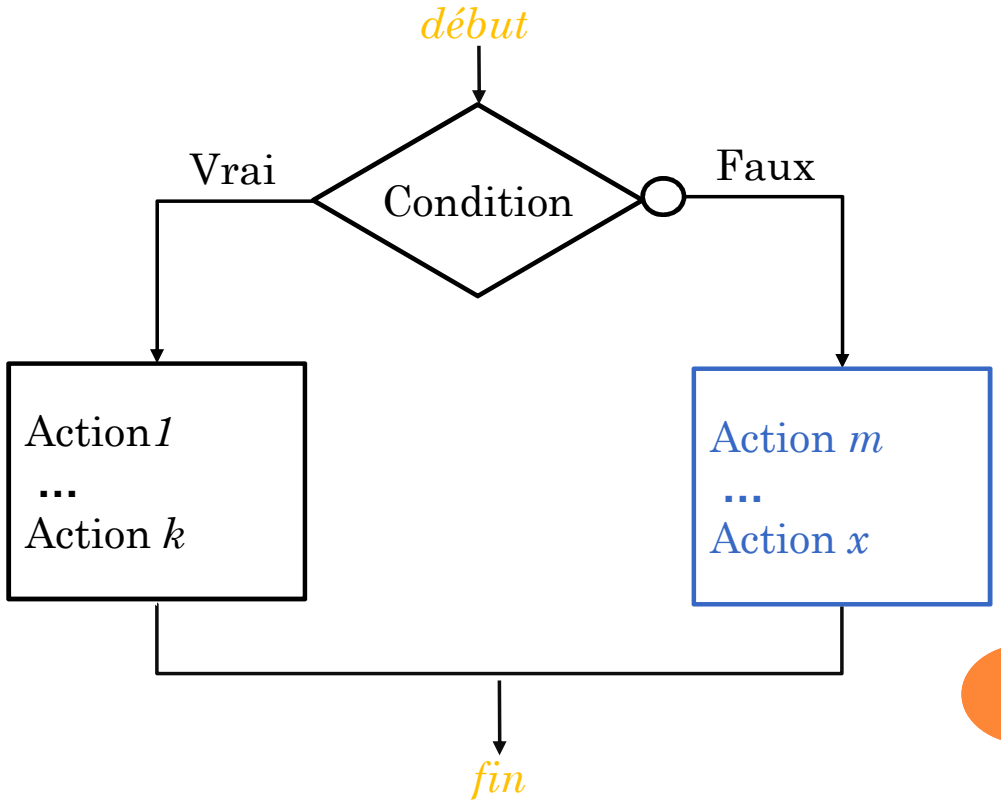
EXEMPLE :

```
DEBUT_PROGRAMME
    |
    | DECLARATIONS
    |       | Entier : x, val
    |       |
    | ACTIONS
    |       |
    |       | ECRIRE_ECRAN  " Hello, faisons une incrémentation ", FDL
    |       | ECRIRE_ECRAN  " Donner une valeur entre 1 et 10 : "
    |       | LIRE_CLAVIER x
    |       | SI ( x>10 ou x < 1) ALORS
    |       |       | ECRIRE_ECRAN « erreur – suite impossible »
    |       |       |
    |       | SINON
    |       |       | val ← x
    |       |       | REPETER
    |       |       |       | val ← val + 1     % incrémentation de val ou accumulation
    |       |       |       | ECRIRE_ECRAN val, FDL
    |       |       |TANT_QUE (val <20)
    |       | FSI
FIN_PROGRAMME
```

Actions structurées – La Sélection

- Si et Suivant
  - Permettent un choix (ou une alternative) entre 2 ou plusieurs possibilités
- Si

début
Si (condition est vraie)
| **Alors**
| | Action *1*
| | ...
| | Action *k*
| **Sinon**
| | Action *m*
| | ...
| | Action *x*
FSI
fin



EXEMPLE :

```
DEBUT_PROGRAMME
    |
    | DECLARATIONS
    |       | Entier : x, val
    |       |
    | ACTIONS
    |       |
    |       | ECRIRE_ECRAN  " Hello, faisons une incrémentation ", FDL
    |       | ECRIRE_ECRAN  " Donner une valeur entre 1 et 10 : "
    |       | Lire-clavier x
    |       | SI (x >10 ou x < 1) ALORS
    |       |     | ECRIRE_ECRAN « erreur – suite impossible »
    |       |     |
    |       | SINON
    |       |     | val ← x
    |       |     | REPETER
    |       |     |       | val ← val + 1     % incrémentation de val ou accumulation
    |       |     |       | ECRIRE_ECRAN val, FDL
    |       |     |TANT_QUE (val <20)
    |       | FSI
FIN_PROGRAMME
```

La Sélection

- Suivant

Suivant (variable)
| **valeur1 :**
| | Action a*1*
| | **…**
| | Action a*n*
| **…**
| **valeurk :**
| | Action *k1*
| | **…**
| | Action *km*
| **Autre cas :**
| | Action *r1*
| | **…**
| | Action *rm*
FSvt

Suivant la valeur de **variable**,
Si **variable** vaut **valeur1** alors faire les actions **a1** à **An**,
Si **variable** vaut **valeurk** alors faire les actions **k1** à **km**
Etc …

Valeur 1..k : entier ou caractère seulement
pas de réel

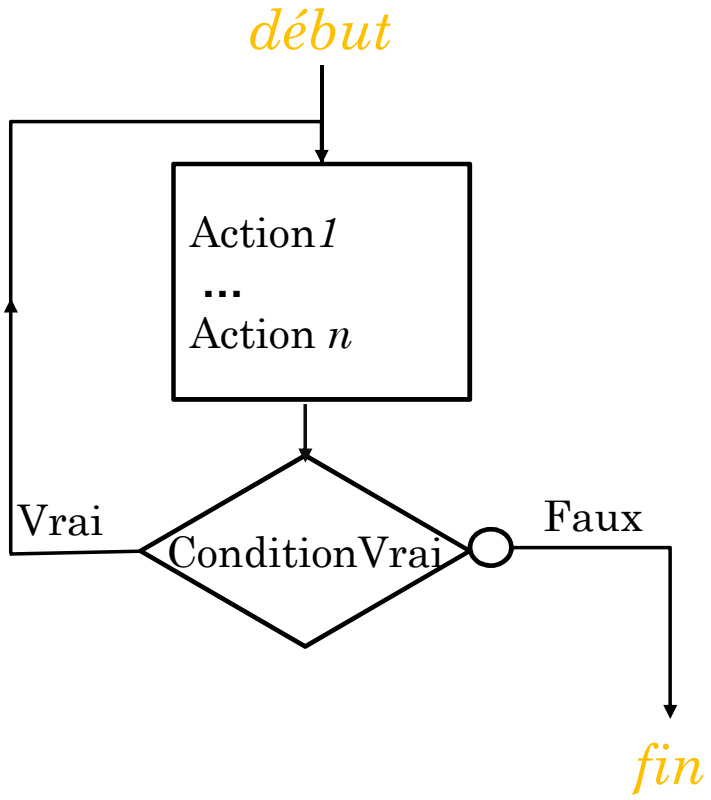
Actions Structurées : Les Boucles

- Boucle
  - Permet la répétition d'actions
  - La boucle s'arrête en fonction de la condition d'arrêt

- 3 sortes de boucles
  - REPETER … TANTQUE ()
  - TANTQUE (…) …. FinTantQue
  - POUR(…) … FinPouR

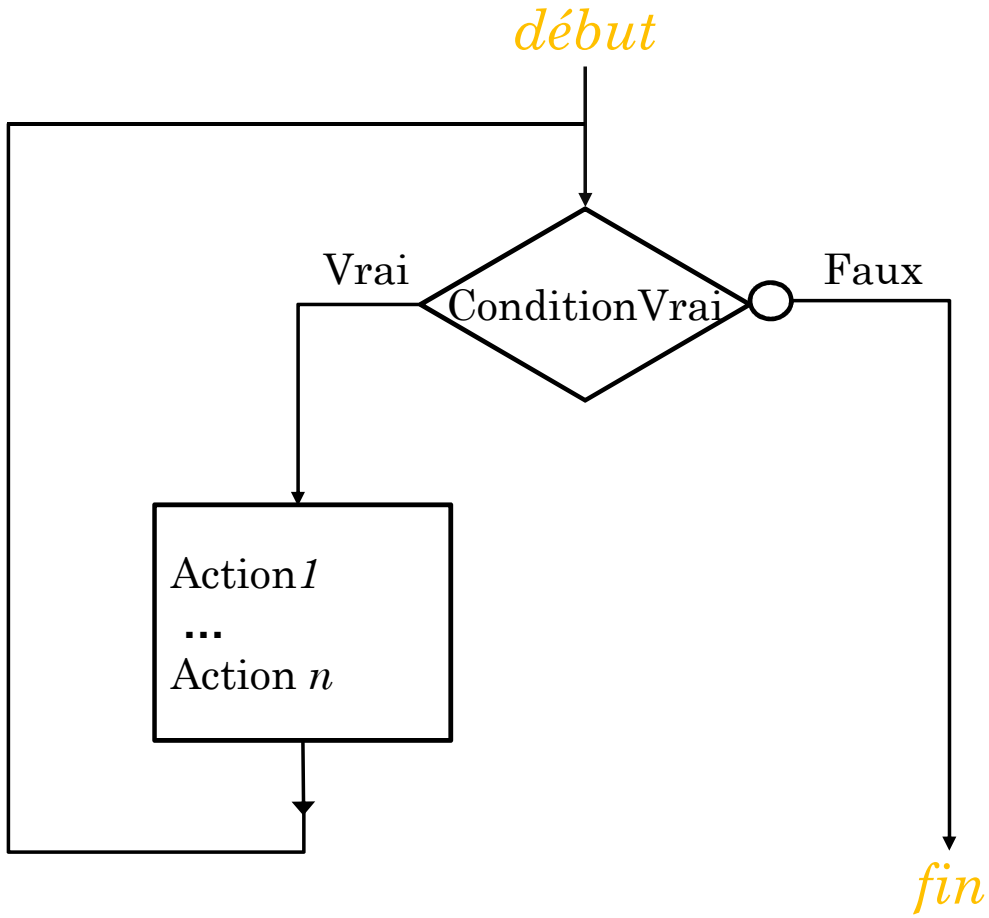
BOUCLES : RÉPÉTER

début
REPETER
| Action *1*
| **...**
| Action *n*
TANTQUE (ConditionVrai)
fin



Boucles : Tant Que

début
TANT_QUE (ConditionVrai)
| Action *1*
| **...**
| Action *n*
FinTQ
fin



Boucles : Pour

début

POUR (initialisation) Quand (Condition) Avec (Progression)

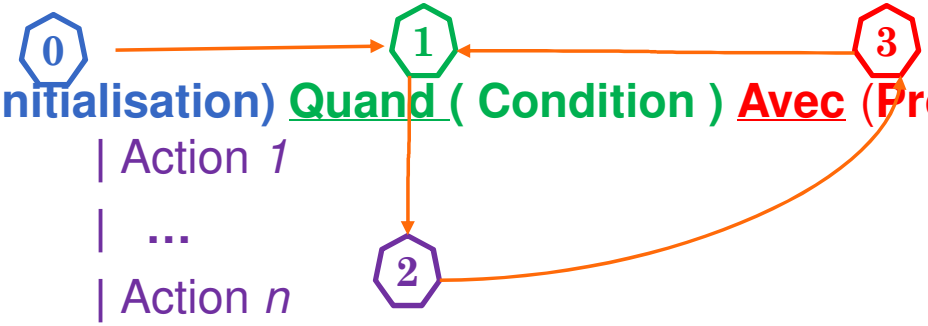
| Action *1*

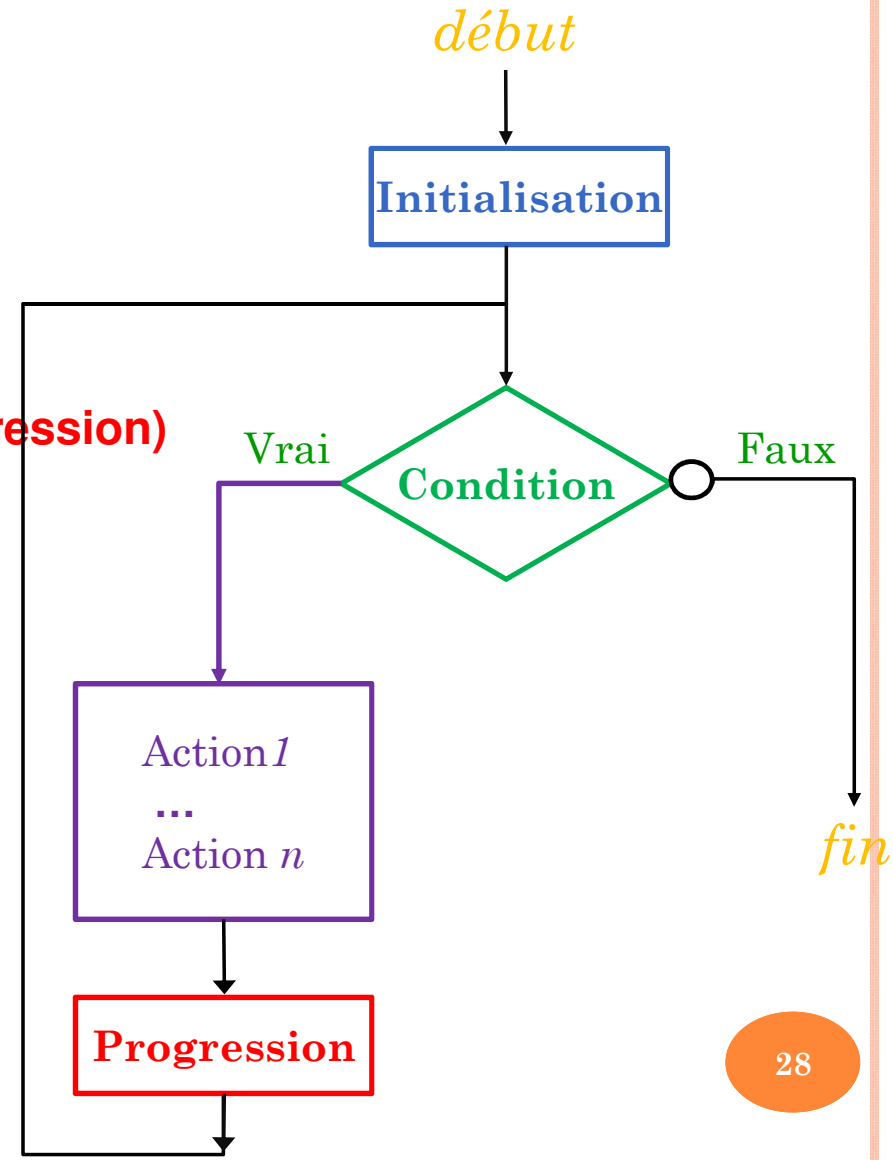
| ...

| Action *n*

FinPouR

fin





EXEMPLE :

```
DEBUT_PGME
    |
    | DECLARATIONS
    |       | Entier : x, val
    |       |
    | ACTIONS
    |       |
    |       | ECRIRE_ECRAN  " Hello, faisons une incrémentation ", FDL
    |       | ECRIRE_ECRAN  " Donner une valeur entre 1 et 10 : "
    |       | Lire-clavier x
    |       | SI (x >10 ou x < 1) ALORS
    |       |     | ECRIRE_ECRAN « erreur – suite impossible »
    |       |     |
    |       | SINON
    |       |     | val ← x
    |       |     | REPETER
    |       |     |       | val ← val + 1     % incrémentation de val ou accumulation
    |       |     |       | ECRIRE_ECRAN val, FDL
    |       |     |TANT_QUE (val <20)
    |       | FSI
FIN_PGME
```

BOUCLES : CRITÈRES DE SÉLECTION

Quelle boucle choisir ?

Critères de choix de la boucle la plus appropriée

1) Peut-on estimer le nombre de répétitions avant son exécution ?

- ☞ OUI ➔ Utilisation de la **POUR**
  - ☞ NON ➔ question suivante

  2) Doit-on exécuter au moins une fois les actions ?
    - ☞ OUI ➔ Utilisation de la **REPETER**
    - ☞ NON ➔ Utilisation de la **TANT_QUE**

Les Boucles – Exemple 1

- Donner l'algo d'un programme qui doit décrémenter de 3 la valeur d'une variable puis l'afficher, tant que cette valeur est supérieure à 20.

La variable réelle est nommée valReel et sa valeur initiale sera lue au clavier (donc donnée par l'utilisateur).

```
DEBUT_PGME
    |
    | DECLARATIONS
    |      | Réel : valReel
    |      |
    | ACTIONS
    |      |
    |      | LIRE_CLAVIER valReel
    |      | TANT_QUE (valReel > 20)
    |      |      | valReel ← valReel - 3
    |      |      |ECRIRE_ECRAN valReel, FDL
    |      | FTQ
FIN_PGME
```

Les Boucles – Exemple 2

- Donner l'algo d'un programme qui doit afficher les valeurs d'une variable entière nommée valEntier variant de 10 à 24 (inclus)

```
DEBUT_PGME
    |
    | DECLARATIONS
    |      | Entier : valEntier

    |      |
    | ACTIONS
    |      |
    |      | POUR (valEntier ←10) QUAND (valEntier <25) AVEC (valEntier ← valEntier + 1)
    |      |    |      | ECRIRE_ECRAN valEntier, FDL
    |      |  FPOUR
    |      |
FIN_PGME
```

Les Boucles – Exemple 3

- Donner l'algo d'un programme qui lit un nombre entier qui doit être compris entre 0 et 20 inclus. Si ce n'est pas le cas, le nombre est lu à nouveau.

  La variable entière est nommée valEnt?

```
DEBUT_PGME
    |
    | DECLARATIONS
    |      | Entier : valEnt
    |      |
    | ACTIONS
    |      |
    |      | REPETER
    |      |      |LIRE_CLAVIER valEnt
    |      | TANT_QUE (valEnt > 20 ou valEnt  <0)
    |      |
FIN_PGME
```

Les Boucles – exemple 4

- Donner l'algo d'un programme qui doit lire au clavier 10 nombres entiers, doubler leur valeur, et afficher à l'écran les nouvelles valeurs (au fur et à mesure de la lecture)

```
DEBUT_PGME
    |
    | DECLARATIONS
    |      | Entier : cpt, nbre
    |      |
    | ACTIONS
    |      |
    |      | ECRIRE_ECRAN  "Donner 10 nombres "   , FDL
    |      | POUR (cpt ← 0) QUAND (cpt <10) AVEC (cpt ← cpt+1)
    |      |      | LIRE_CLAVIER nbre
    |      |      | nbre ← nbre *2
    |      |      | ECRIRE_ECRAN nbre, FDL
    |      | FPR
    |      |
FIN_PGME
```

Les Boucles – Exemple 5

- Donner l'algo d'un programme qui doit lire 10 nombres entiers, en faire la somme puis afficher le résultat final.

```
DEBUT_PGME
    |
    | DECLARATIONS
    |       | Entier : cpt, nbre, somme
    |       |
    | ACTIONS
    |       |  somme ← 0
    |       | ECRIRE_ECRAN " Donner 10 nombres " , FDL
    |       | POUR (cpt ← 0) QUAND (cpt <10) AVEC (cpt ← cpt +1)
    |       |       | LIRE_CLAVIER nbre
    |       |       |somme ← somme + nbre
    |       | FPR
    |       | ECRIRE_ECRAN somme, FDL
FIN_PGME
```

Les Boucles – Exemple 6

- Donner l'algo d'un programme qui doit lire 10 nombres réels et doit vérifier que chaque nombre lu soit compris entre 0 et 20 inclus. Si le nombre lu est en dehors de ces bornes, la lecture du nombre est répétée.

```
DEBUT_PGME
    |
    | DECLARATIONS
    |       |entier : cpt
    |       | reel : nbre
    |       |
    | ACTIONS
    |       | ECRIRE_ECRAN " Donner 10 nombres compris entre 0 et 20" , FDL
    |       | POUR (cpt ← 0) QUAND (cpt <10) AVEC (cpt ← cpt +1)
    |       |       | REPETER
    |       |       |    | lire-cavier nbre
    |       |       | TANT QUE (nbre <0 ou nbre >20)
    |       | FPR
    |       | FIN_PGME
```

TRACE D'UN ALGORITHME

- Une trace est un exemple d'exécution qui indique l'évolution des valeurs des variables.
- Ainsi la valeur des variables concernées est modifiée après chaque action
- Exemple :

```
DEBUT_PGME
    |
    | DECLARATIONS
    |       | entier : cpt, nbre, somme
    |       |
    | ACTIONS
    |       | somme ← 0
    |       | nbre ← 1
    |       | POUR (cpt ← 0) QUAND ( cpt<5) AVEC ( cpt ← cpt+1)
    |       |       | nbre← nbre*2
    |       |       |somme ← somme +nbre
    |       |       |somme ← somme +1
    |       |       |ecrire-ecran somme, FdL
    |       | FPR
FIN_PGME
```