INFO0101

INTRODUCTION À L'ALGORITHMIQUE ET À LA PROGRAMMATION

COURS 1

PRÉSENTATION DU MODULE, ORGANISATION ET NOTIONS DE BASE

Pierre Delisle, Cyril Rabat, Christophe Jaillet, Jessica Jonquet et François Alin
Département de Mathématiques et Informatique
Septembre 2017

Plan de la séance

- Description du cours
  - Objectifs, organisation
- Préliminaires
  - Fonctionnement d'un ordinateur
- Introduction à l'algorithmique
  - Définition d'un algorithme
  - Variables, types
  - Opérateurs et expressions
  - Instructions élémentaires/structurées

Objectifs du module

- Acquérir les notions et principes de base
  - En algorithmique
  - En programmation
- Apprendre à écrire des programmes en Java

Organisation

- CMTDi
  - 20 x 2 heures
- TP
  - 10 x 2 heures (en demi-groupes)
  - Début semaine 37
- **Surveillez les emplois du temps !**
- **CMTDi et TP obligatoires !**
- **Activation de votre compte avant le 1er TP !**
  - Vous recevrez un mail des ingénieurs responsables vous expliquant la procédure à suivre...

Évaluation

Nature de l'évaluation	Nombre de points
Devoir Surveillé 1	20
Devoir Surveillé 2	20
Devoir Surveillé Terminal	30
Tp-Test	20
Compte-rendu de TP	10
Total	100

- Licence INFO/MATH → Total du Semestre 1 : 500 points
  - INFO0101 : 100 points
  - MA0101 : 100 points
  - PH0101 : 100 points
  - MA0102 : 50 points
  - EL0103 : 50 points
  - MOI0101 – METH0101 – PPRO0101 – AN0101 : 100 points

Quelques informations pratiques...

- Emplois du temps et informations
  - Bureau virtuel
    - ebureau.univ-reims.fr/
  - Site Web de la Licence Informatique
    - http://cosy.univ-reims.fr/LicenceINFO/
  - Site Web de la scolarité
    - http://www.univ-reims.fr/sciences
- Demi-groupes de TP
  - Constitution d'ici le premier TP
- Sites Web de cours – Supports et énoncés de TD
  - http://cosy.univ-reims.fr/~pdelisle/
- Site des TP
  - http://www.cyril-rabat.fr/enseignement/Info0101/

Structure du module

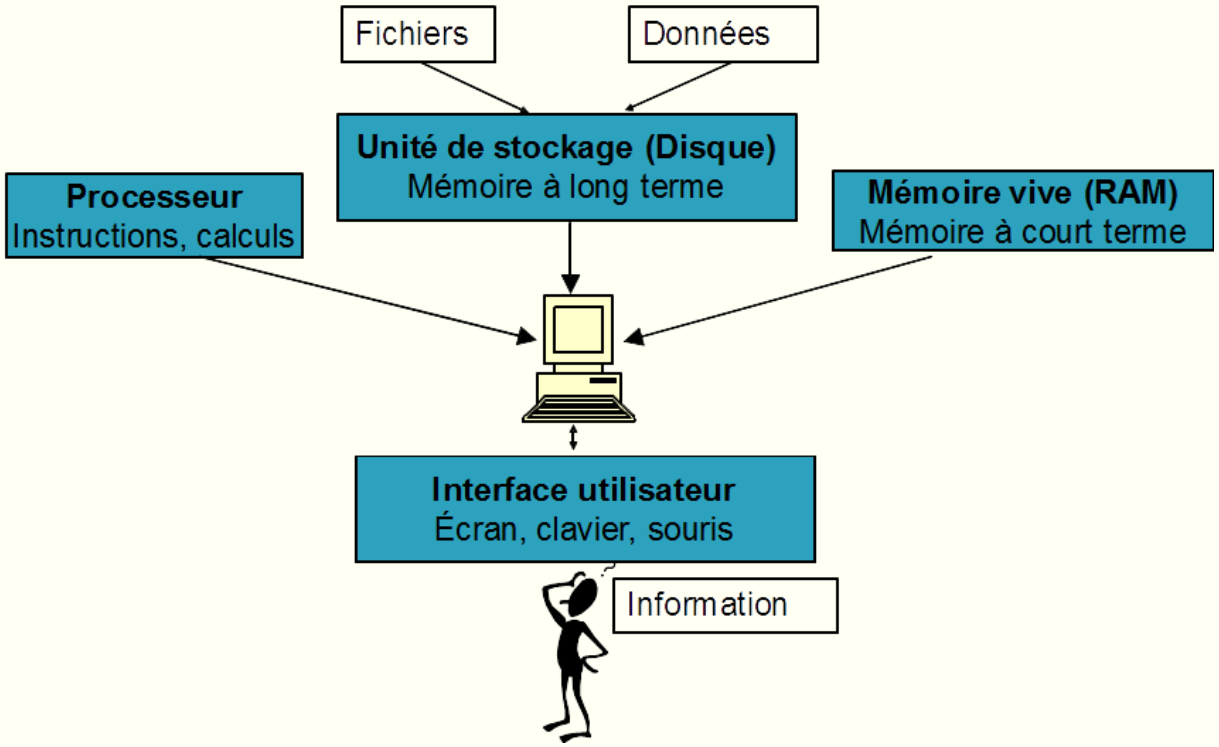
- Introduction
  - Ordinateur, données, traitements …
- Représentation des données
  - Types
  - Variables
- Algorithmique
  - Instructions simples: affectation, arithmétique, …
  - Instructions structurées: conditions, boucles, …
  - Fonctions et procédures
  - Tableaux
- Si le temps le permet...
  - Tableaux à 2 dimensions
  - Stockage des données - Fichiers
  - Compléments d'algorithmique
- Mise en œuvre : Langage Java

PRÉLIMINAIRES

Fonctionnement d’un ordinateur

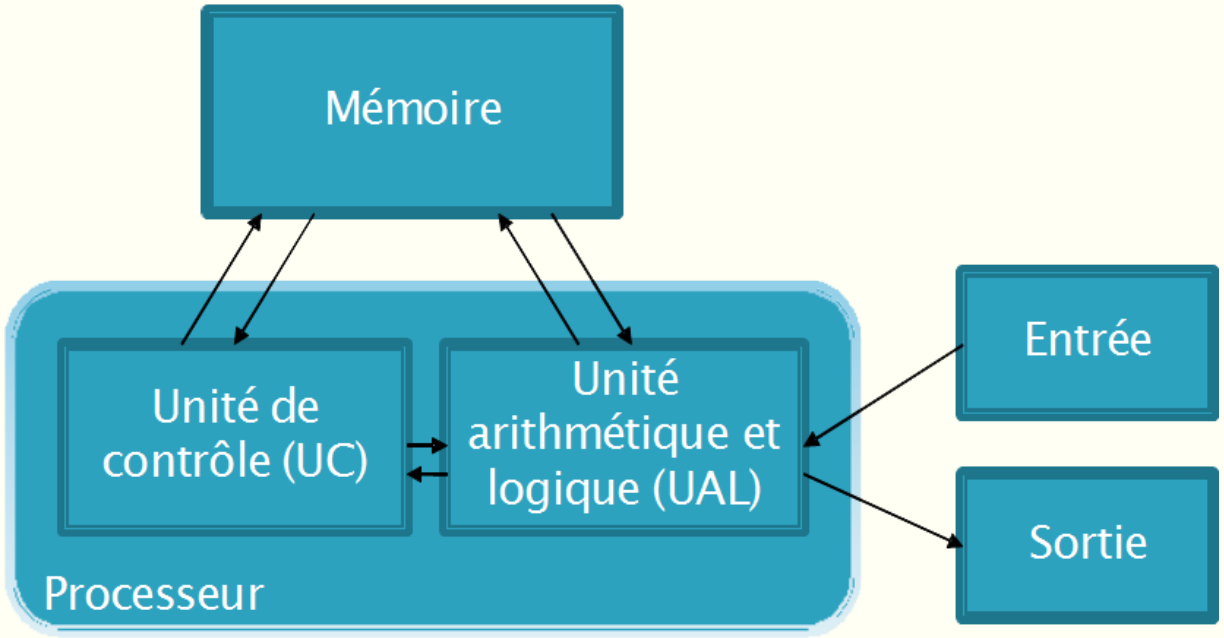
Qu'est-ce qu'un ordinateur ?

- Machine capable
  - d'exécuter automatiquement et fidèlement une série d'**opérations simples** qu'on lui a indiquée
  - de manipuler rapidement et sans erreur un grand nombre d'informations
- Pour résoudre un problème à l'aide d'un ordinateur, il faut :
  - Analyser ce problème
  - Déterminer une méthode de résolution
    - Suite des opérations à effectuer (algorithme) pour obtenir la solution
  - Traduire l'algorithme dans un langage de programmation adapté



La structure d'un ordinateur

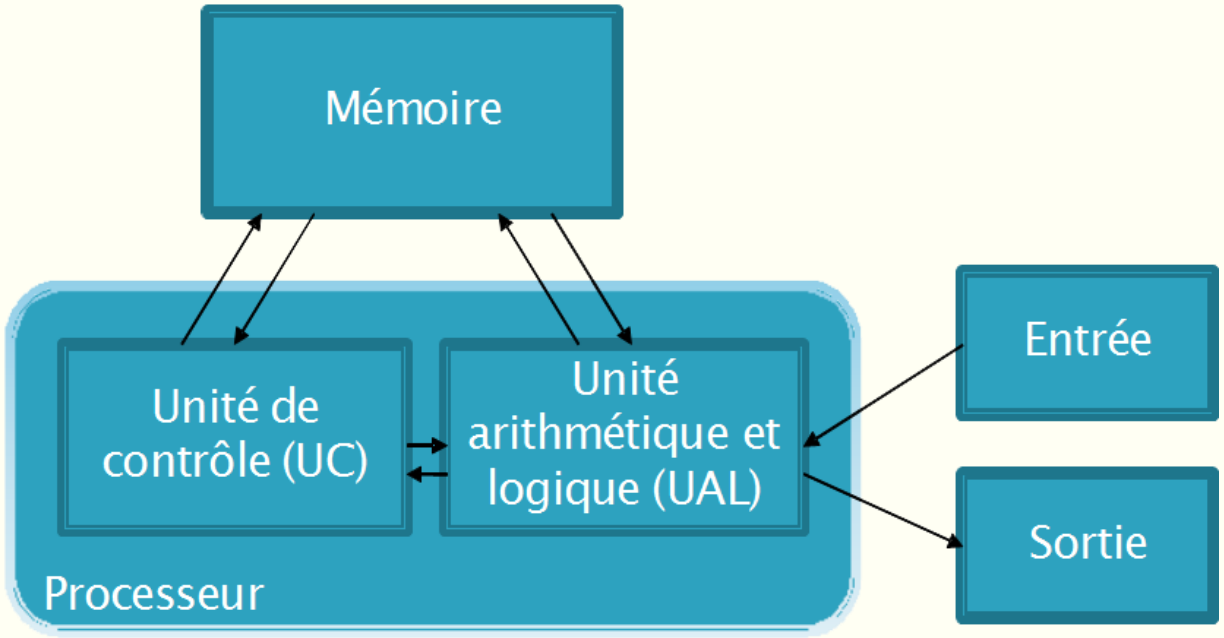
- Basé sur le modèle de Von Neumann



- Le processeur
  1) Charge une instruction à exécuter
  2) Décode l'instruction
  3) Localise dans la mémoire les données utilisées par l'instruction
  4) Charge les données si nécessaire
  5) Exécute l'instruction
  6) Sauvegarde les résultats à leurs destinations respectives
  7) Passe à l'instruction suivante

La structure d'un ordinateur

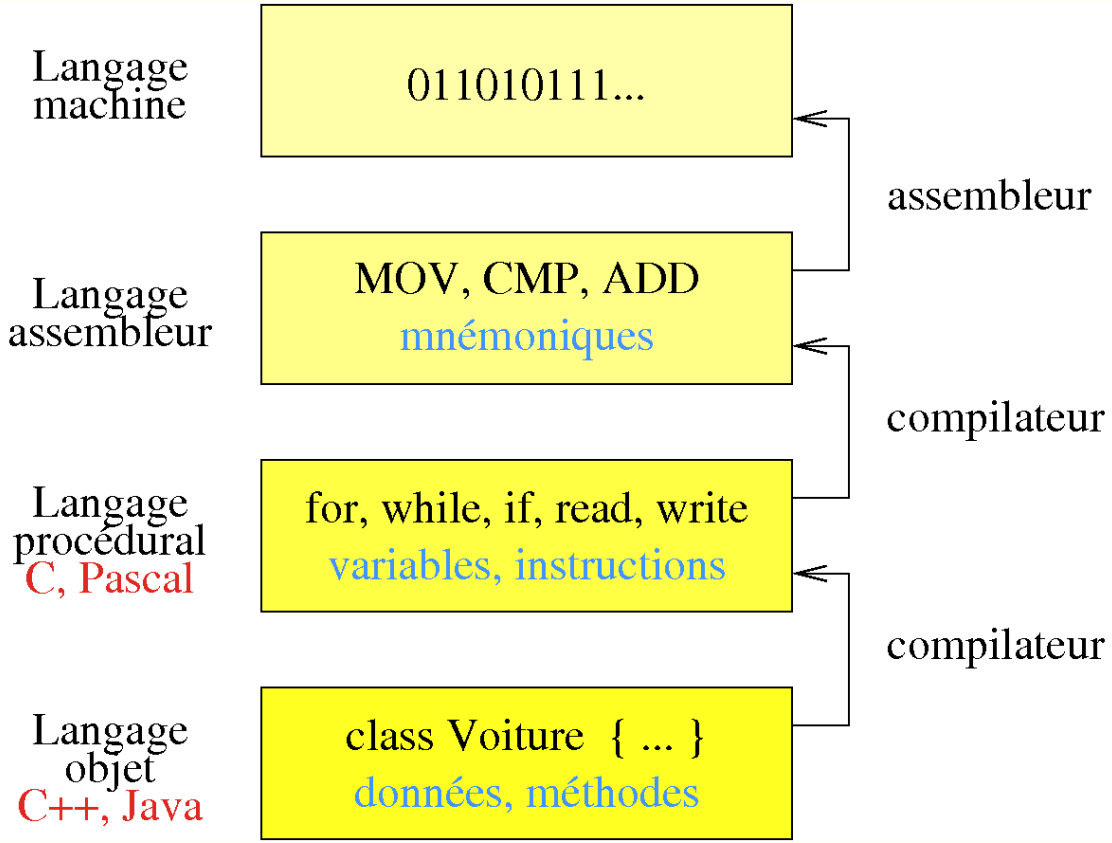
- Basé sur le modèle de Von Neumann



- La mémoire
  - Stocke les données et les instructions
  - Le processeur peut y accéder à n'importe quel moment
    - En lecture : consultation
    - En écriture : modification
  - Formée d'un certain nombre de cellules, ou cases, contenant chacune une information
- Les entrées/sorties
  - Échange avec l'environnement externe (utilisateur)
  - Périphériques de communication
    - Clavier
    - Souris
    - Imprimante
    - …

Objectif général du cours

- Algorithmique
  - Décomposer un problème en une suite d'instructions simples pouvant être exécutées par un ordinateur (théorique)
  - L'ordinateur n'est pas intelligent, il faut lui dire quoi faire dans un langage qu'il comprend !
  - Formalisme indépendant de tout langage de programmation
- Programmation
  - Écrire un algorithme dans un langage de programmation spécifique afin de pouvoir l'exécuter sur un ordinateur réel



INTRODUCTION À L'ALGORITHMIQUE

Plan

1) Algorithme : définitions
2) Informations élémentaires
   - Variables
   - Constantes
3) Types de données élémentaires
   - Entier
   - Flottant
   - Caractère
   - Booléen
4) Instructions élémentaires
   - Affectation
   - Lecture
   - Écriture
   - Génération d'un nombre aléatoire
5) Présentation d'un algorithme
6) Opérateurs et expressions
   - Arithmétiques
   - Logiques
   - Relationnels
7) Instructions structurées
   - Si alors … Sinon
   - Cas Parmi
   - Pour
   - Tant que
   - Faire … Tant que

Algorithmique ?

- Le problème principal du concepteur ou du programmeur
  - À partir des informations fournies, décrire la suite des actions élémentaires permettant de résoudre un problème et de produire les résultats attendus
- Algorithmique
  - Étude et production de règles et techniques impliquées dans la conception d'algorithmes
- Algorithme (Encyclopaedia Universalis)
  - Suite finie de règles à appliquer dans un ordre déterminé à un nombre fini de données pour arriver, en un nombre fini d'étapes, à un certain résultat
- Le rôle de l'algorithme est fondamental en informatique
  - Sans algorithme, pas de programme
  - Un programme n'est que sa traduction dans un langage compréhensible par l'ordinateur
- Un algorithme est indépendant à la fois
  - Du langage de programmation dans lequel il est traduit
  - De l'ordinateur sur lequel le programme doit être exécuté

Les informations élémentaires manipulées par un algorithme

- Une information élémentaire est désignée par
  - Un nom (invariable) -> pour l'utiliser
  - Un type (invariable) -> pour l'interpréter
  - Une valeur
    - Modifiable (variable)
    - Non modifiable (constante)

Nom	Type	Valeur
n	Entier	3

- Section de déclarations
  - Permet de définir les informations élémentaires nécessaires au début de l'algorithme
- Exemple 1
  - Algorithme Info0101
- Les algorithmes doivent pouvoir manipuler des données de différentes natures
  - Pour les représenter adéquatement, il faut connaître les **types de données** élémentaires

LES TYPES DE DONNÉES ÉLÉMENTAIRES

Pourquoi les types de données ?

- Ordinateur -> Langage binaire
  - Ne reconnaît que les suites de bits (0 et 1)
- La même suite de bits peut représenter
  - Une instruction
  - Une donnée (nombre, caractère, etc.)
- Exemple : 01100001
  - Si interprétée comme un entier : 97
  - Si interprétée comme un caractère (ascii) : ‘a’
- Le typage d’une variable permet d’interpréter sa valeur, de lui donner un sens

Les principaux types de données

- Entier
  - Permet de représenter des valeurs numériques entières positives ou négatives

    0, 12, 150, -43, …

```
Déclarations
  Variables
    a : entier
```

- Flottant
  - Permet de représenter (de manière approchée) une partie des nombres réels
  - Notation décimale

    1,346 6567,4552 -4,0
  - Notation exponentielle

    1346E-3 6,5674552e3 -4e0

```
Déclarations
  Variables
    a : réel
```

- Caractère
  - Permet de représenter un seul caractère
  - Noté entre apostrophes

    ‘a’, ‘i’, ‘ ‘, ‘+’, ‘Z’, …

```
Déclarations
  Variables
    a : caractère
```

- Booléen
  - Représente une valeur logique de type vrai/faux
  - Ne peut prendre que 2 valeurs
    - Vrai (ou 1)
    - Faux (ou 0)

```
Déclarations
  Variables
    a : booléen
```

- Afin de manipuler les données dans un algorithme, il faut utiliser des **instructions**

LES INSTRUCTIONS ÉLÉMENTAIRES

Affectation, Lecture et Écriture

Affectation

- Attribuer une valeur donnée à une variable
- Notation

```
variable ← valeur
```

- La valeur et la variable doivent être du même type
- Après une affectation, l'ancienne valeur de la variable est perdue
- La valeur peut être le résultat d'une expression

- Exemple

```
Déclarations
  Variables
    a : entier
    b : caractère
Début
  a ← 2
  b ← 'a'
Fin
```

- Attention aux variables non initialisées !

```
a ← 2
b ← 2 + a
a ← a + 1
a ← 2 * b + e
```

Valeur de e ???

- Exemple 2 : échange des valeurs de 2 variables

Lecture

- Attribuer une valeur à une variable par l'intermédiaire d'un périphérique d'entrée
  - Clavier, souris, etc.
- Notation

```
variable1 ← lire ()
variable2 ← lire ("Entrez une valeur")
```

- Proche de l'affectation, mais la valeur est à l'initiative de l'utilisateur

- Exemple

```
a ← 2
a ← lire ()
```

Écriture

- Imprimer (afficher) une valeur ou le contenu d'une variable sur un périphérique de sortie
  - Écran, imprimante, etc.
- Notation

```
écrire (variable)
écrire (variable1 + variable2)
écrireLn (variable1 + "  et " + variable2)
```

- Ne modifie pas la valeur de la variable

- Exemple

```
a ← 2
écrire ("Entrez une valeur")
b ← lire()
c ← a + b
écrire (c)
```

Génération d'un nombre aléatoire

- Générer un nombre réel au (pseudo) hasard dans l'intervalle [0, 1 [
- Notation

```
variable ← aleatoire()
```

- La variable prendra une valeur aléatoire réelle entre 0 et 0,999999…
- Exemple

```
Déclarations
  Variables
    val : réel
Début
  val ← aleatoire()              /* val prend une valeur aléatoire réelle dans l'intervalle [ 0, 1 [ */
  val ← aleatoire() * 10         /* val prend une valeur aléatoire réelle dans l'intervalle [ 0, 10 [ */
  val ← aleatoire() * 10 - 5     /* val prend une valeur aléatoire réelle dans l'intervalle [ -5, 5 [ */
Fin
```

Présentation d'un algorithme

- Expliciter les actions principales par des commentaires entre /* et */
- Mettre les mots clés en évidence et utiliser une indentation qui facilite la lecture

```
Algorithme Échange
Déclarations
  Variables
    a, b, temp : entier
Début
  a ← lire()            /*initialisation de a*/
  b ← lire()            /*initialisation de b*/
  temp ← a              /*sauvegarde de la valeur de a*/
  a ← b                 /*recopie de la valeur de b dans a*/
  b ← temp              /*recopie de l'ancienne valeur de a dans b*/
  écrire(a + " " + b)   /*affichage des valeurs de a et b*/
Fin
```

OPÉRATEURS ET EXPRESSIONS DE BASE

Notion d'opérateur et d'expression
Opérateurs arithmétiques, relationnels et logiques

Opérateur et expression

- Les opérateurs (vus plus loin)

  +, -, *, …

- Sont utilisés dans des expressions

  x + b

- Qui sont elles-mêmes utilisées dans des instructions

  $y \leftarrow a * x + b$

- Qui, combinées de différentes façons, constitueront un algorithme/programme

Opérateurs arithmétiques

- Opérateurs binaires (2 opérandes)

Addition	+
Soustraction	-
Multiplication	*
Division	/
Modulo	%

- Opérateurs unaires (1 seul opérande)

Opposé	-
Identité	+

- Fonctionnement des opérateurs binaires
  - A priori définis pour 2 opérandes de même type, fournissent un résultat de ce type
  - Les conversions implicites permettent éventuellement de spécifier des opérandes de types différents
- Priorités relatives des opérateurs
  1) Opérateurs unaires + et –
  2) Opérateurs binaires *, / et %
  3) Opérateurs binaires + et -
- Priorité identique : calcul de gauche à droite

Opérateurs arithmétiques - Exemple

```
Déclarations
  Variables
    a, b, c, d, e : entier
Début
  a ← 1            /*initialisation de la variable a*/
  b ← 2            /*initialisation de la variable b*/
  c ← a + b        /*c vaut 3 : 1 + 2*/
  d ← a + b * c  /*d vaut 7 : 1 + (2 x 3)*/
  e ← -b          /*e vaut -2 : opposé de 2*/
Fin
```

Opérateurs relationnels

- A priori définis pour des opérateurs de même type, mais aussi soumis aux conversions implicites
- Retourne une valeur booléenne (vrai/faux)

Opérateur	Signification
<	Inférieur à
<=	Inférieur ou égal à
>	Supérieur à
>=	Supérieur ou égal à
==	Égal à
!=	Différent de

- Exemple

```
Déclarations
  Variables
    a, b : entier
    c, d : booléen
Début
  a ← 1          /*initialisation de la variable a*/
  b ← 2          /*initialisation de la variable b*/
  c ← a == b     /*c vaut                          */
  d ← a < b      /*d vaut                          */
Fin
```

Opérateurs logiques

- Retourne un booléen (Vrai ou Faux)
- Permet de combiner plusieurs expressions
- ET
  - Vrai si et seulement si les 2 opérandes sont vraies
- OU
  - Vrai si au moins un des deux opérandes est vrai
- NON
  - Vrai si l'opérande est faux
  - Faux si l'opérande est vrai

- Exemple

```
Déclarations
  Variables
    a, b : entier
    c, d : booléen
Début
  a ← 1                          /*initialisation de la variable a*/
  b ← 2                          /*initialisation de la variable b*/
  c ← (a == b) ET (b > 0)        /*c vaut [          ]
  d ← c OU (a > 0)               /*d vaut [          ]
Fin
```

- Utiliser des parenthèses pour spécifier l'ordre des opérations et/ou clarifier l'algorithme

INSTRUCTION STRUCTURÉES

Structures de sélection (conditionnelles) : Si Alors Sinon, Cas Parmi
Schémas itératifs (boucles) : Pour, Tantque, Faire Tantque

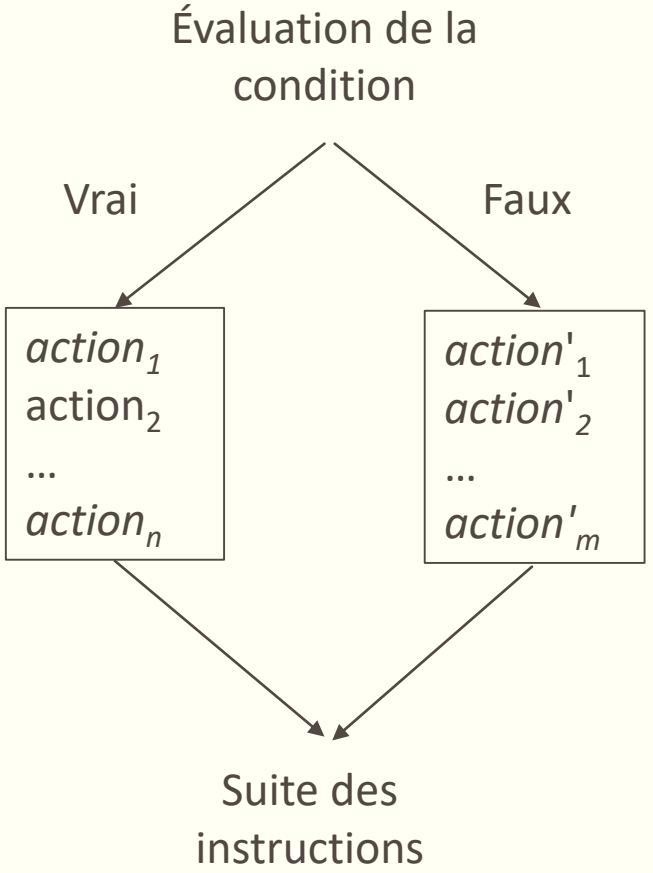
Structures de sélection (conditionnelles)

- Expriment la notion de choix
- Branchement : l'évaluation d'une condition déterminera quelles instructions seront effectuées
- Condition : expression booléenne

Si … Alors … Sinon

- Si la condition est vraie, on exécute une suite d'instructions
- Si la condition est fausse, on exécute une autre suite d'instructions
- Notation

Si **condition** Alors
  $action_1$
  $action_2$
  …
  $action_n$
Sinon
  $action'_1$
  $action'_2$
  …
  $action'_m$
FinSi



- La partie Sinon est facultative
  - S'il n'y pas de Sinon, FinSi est placé après le premier groupe d'actions

Exemples

Exemple 3 : Calcul du salaire d'un fonctionnaire

- On veut calculer le salaire brut d'un employé en fonction de
  - Son taux horaire
  - Le nombre d'heures de travail
- Si l'employé fait plus de 100 heures dans le mois, on lui rajoute une prime
- S'il ne fait pas plus de 100 heures, on lui ajoute une prime de sous-emploi

Exemple 4 : Maximum de 2 valeurs

- On veut connaître la valeur maximum de deux valeurs entrées au clavier par l'utilisateur

Cas … Parmi

- Quand un ensemble de conditions teste les différentes valeurs possibles d'une même expression ou variable, on peut utiliser la structure *Cas Parmi*

```
Cas Expression Parmi
  valeur1 : action1
  valeur2 : action2
  valeurn : actionn
  défaut :  actionn+1   /*optionnel*/
FinCas
```

- Exemple 5 – Prix d'un billet de train après réduction
  - On veut connaître le prix du billet de train d'un client après réduction
  - Soit le voyageur est abonné : 50%
  - Soit le voyageur est un enfant de - de 5 ans : 100%
  - Soit le voyageur est une personne âgée : 60%

Feuille de TD # 1

- Exercice 1

Avant d'aller plus loin : Trace d'exécution d'un algorithme

- Permet de suivre pas-à-pas le déroulement de l'algorithme
  - Meilleure compréhension du fonctionnement
  - Permet éventuellement de trouver les erreurs (bugs)
- Il faut expliciter
  - Le numéro des instructions
  - L'évaluation de contrôle, si applicable (conditions, boucles, appels de fonction/procédure)
  - La valeur des variables (état initial et changements)
  - L'affichage

Inst.	*Contrôle*	*Variables locales*			
		a	*b*	*c*	*resultat*
Avant		*?*	*?*	*?*	*?*
1	×	*-4*			
2	×		*3*		
3	×			*-1*	
4	×	*4*			
5	×			*2*	
6	*(a>c) ? Vrai*				
6a.1	×		*7*		
7	×				*17*

Exemple 6 : Trace d'exécution de l'algorithme de l'exemple 3

```
                Algorithme SalaireFonctionnaire
                Déclarations
                  Variables
                     nbHeures                                  : entier
                     tauxHoraire, salaireBrut, prime, primeSE  : réel
                Début
{1}                 écrire("Entrez le taux horaire : ")
{2}                 tauxHoraire ← lire ()
{3}                 écrire("Entrez le nombre d'heures de travail : ")
{4}                 nbHeures ← lire ()
{5}                 salaireBrut ← tauxHoraire * nbHeures
{6}                 Si nbHeures > 100 Alors
{6a.1}                 écrire("Entrez la prime : ")
{6a.2}                 prime ← lire ()
{6a.3}                 salaireBrut ← salaireBrut + prime
                    Sinon
{6b.1}                 écrire("Entrez la prime de sous-emploi : ")
{6b.2}                 primeSE ← lire ()
{6b.3}                 salaireBrut ← salaireBrut + primeSE
                    FinSi
{7}                 écrire("Salaire brut : " + salaireBrut)
                Fin
```

Feuille de TD # 1

- Traces d'algorithmes : Exercice 2
- Synthèse : Exercices 3 à 8

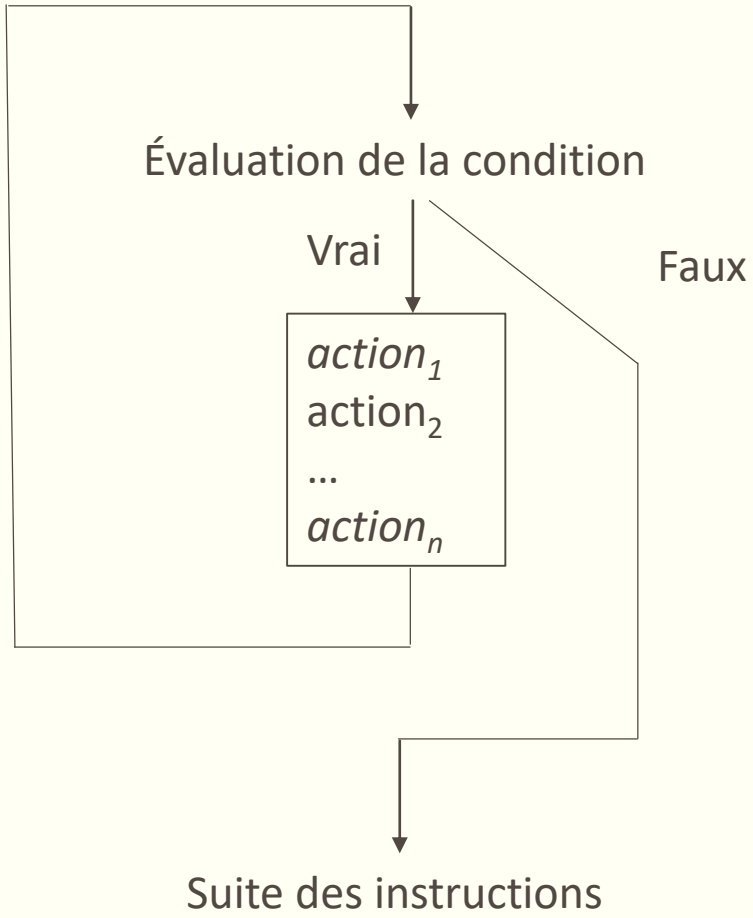
Structures itératives (Boucles)

- Une structure itérative permet de répéter une action ou une séquence d'actions plusieurs fois
- 3 types de boucles
  - TantQue
  - Pour
  - Faire … Tantque

Boucle Tantque

- Permet l'exécution d'une suite d'actions tant qu'une condition de continuité est vérifiée
- On l'utilise lorsqu'on ne peut pas prédire le nombre d'itérations de la boucle
- Notation

```
TantQue condition Faire
   action1
   action2
   …
   actionn
FinTantQue
```



- Condition de continuité recalculée à chaque tour de boucle
- Dès que la condition est fausse, la boucle se termine

Exemples – Boucle TantQue

Exemple 7

- Calculer le maximum d'une suite quelconque d'entiers positifs entrés au clavier, la dernière valeur étant nulle
- Trace d'exécution

Exemple 8

- Écrire un algorithme qui calcule le PGCD de 2 nombres par la méthode des différences

Feuille TD # 2

- Exercice 1.1 – Algorithme 1

Boucle Pour

- Avant chaque passage éventuel dans la boucle, on teste la valeur de la variable de contrôle : on n'effectue les actions de la boucle que si elle n'a pas dépassé $valeur_2$
- Après chaque passage dans la boucle, la variable de contrôle est automatiquement incrémentée de 1
- Après incrémentation, le programme retourne tester si variable_de_contrôle dépasse $valeur_2$
- La variable de contrôle est généralement entière
- $valeur_1$ et $valeur_2$ doivent être de même type que la variable de contrôle

```
Pour variable_de_contrôle allant de valeur1 à valeur2 Faire
  action1
  action2
  …
  actionn
FinPour
```

Boucle Pour

- Si valeur$_1$ > valeur$_2$, la variable de contrôle a dès le départ une valeur supérieure à valeur$_2$, donc la suite d'instructions n'est pas exécutée (même pas une fois)
- À la fin d'une boucle pour, la valeur de la variable de contrôle est indéterminée. Il ne faut pas l'utiliser sans réinitialisation
- Il ne faut pas modifier la valeur de la variable de contrôle à l'intérieur de la boucle
- On utilise une boule pour lorsqu'on sait exactement combien d'itérations on doit réaliser

```
Pour variable_de_contrôle allant de valeur1 à valeur2 Faire
  action1
  action2
  …
  actionn
FinPour
```

Boucle Pour

- Il est parfois utile de faire varier la variable de contrôle par valeurs décroissantes ou par incréments différents de l'unité. Dans ce cas, il faut utiliser la forme la plus générale de la boucle pour

```
Pour variable_de_contrôle allant de valeur1 à valeur2 par incr Faire
  action1
  action2
  …
  actionn
FinPour
```

Exemple 9 : Boucle Pour

- Calcul du maximum de 4 valeurs entrées au clavier
- Trace d'exécution

Feuille TD # 2

- Exercice 1.1 – Algorithme 2

Boucle Faire … Tantque

- Utilisé lorsque le nombre de passages est inconnu (comme pour la boucle TantQue)
- Mais le corps de la boucle est toujours exécuté au moins une fois
- Notation

```
Faire
  action1
  action2
  …
  actionn
TantQue condition     /* La condition doit être fausse */
                       /* pour sortir de la boucle */
```

- L'expression logique est évaluée après l'exécution du corps de la boucle
  1) Le corps de la boule est exécuté
  2) L'expression logique est évaluée
  3) Si sa valeur est vraie
     1) Le corps de la boucle est exécuté à nouveau
     2) L'expression logique est réévaluée
  4) Si sa valeur est fausse, on exécute l'instruction qui suit Tantque

Exemple 10 : Boucle Faire … Tantque

- Saisie au clavier d'une valeur supérieure à 10
- On redemande la saisie jusqu'à ce que la valeur entrée soit valide

Feuille TD # 2

- Exercices 2-6

PROCHAIN COURS :

LE LANGAGE JAVA