Merci à Louise Laforest pour son aide à compléter ces notes de cours.

1 Introduction

Si on veut donner l'ordre de *"monter un escalier composé par trois marches"* à une personne, on va utiliser la langue que cette personne comprend (le français par exemple) et on va lui dire : "Monte l'escalier". Ici on parle d'une instruction (ou d'un *ordre*).

On peut aussi lui dire :

1. "Monte la première marche."
2. "Monte la deuxième marche."
3. "Monte la troisième marche."

Ou :

1. "Lève le pied droit."
2. "Pose le pied droit sur la première marche."
3. "Lève le pied gauche."
4. "Pose le pied gauche sur la première marche."
5. "Lève le pied droit."
6. "Pose le pied droit sur la deuxième marche."
7. "Lève le pied gauche."
8. "Pose le pied gauche sur la deuxième marche."
9. "Lève le pied droit."
10. "Pose le pied droit sur la troisième marche."
11. "Lève le pied gauche."
12. "Pose le pied gauche sur la troisième marche."

Ou encore :

1. "Saute vers la troisième marche."

Nous avons là quatre algorithmes (solutions) qui permettent de résoude le même problème (arriver en haut de l'escalier). La différence entre ces algorithmes est dans le niveau de précision et dans l'approche utilisée.

Un algorithme est un ensemble d'instructions permettant de décrire une solution à un problème.

Dans ce cours nous allons apprendre à donner des instructions (ou des ordres) à des ordinateurs. Pour ça, nous avons besoin d'apprendre un langage que les ordinateurs peuvent comprendre. Dans ce cours, nous apprendrons le langage Java.

Avant d'apprendre Java, nous allons apprendre le *Pseudo-code* qui est un langage à mi-chemin entre le français et un langage de programmation évolué. L'avantage du pseudo-code est qu'il est indépendant du langage de programmation choisi pour mettre en oeuvre la solution.

De plus, on peut facilement traduire le Pseudocode en n'importe quel langage de programmation évolué.

Les principales instructions que nous pouvons exprimer en Pseudocode sont :

- La mémorisation d'une valeur dans un espace mémoire (on parle d'affectation de valeur dans une variable)
- La saisie d'une valeur à travers un clavier
- L'affichage d'une valeur à l'écran
- Les instructions de sélection (ou conditionnelles)
- Les instructions de boucle (ou répétitions)

2 Affectation

Généralement, nous avons besoin d'effectuer des traitements intermédiaires avant d'afficher des résultats. Ces traitements nécessitent de stocker des valeurs intermédiaires dans la mémoire de l'ordinateur.

Pour accéder à la mémoire de l'ordinateur, on utilise la notion de *variables*. Une variable correspond à un espace mémoire identifié par un nom (le nom de la variable) dans lequel on peut stocker des valeurs numériques ou textuelles.

Pour donner l'ordre à l'ordinateur de stocker une valeur dans un espace mémoire, on va utiliser l'instruction suivante :

```
<nom de variable> ← <expression>
```

L'espace mémoire dans lequel la valeur de l'expression est stockée est identifié par le nom de la variable.

3 Saisie des données

Un utilisateur d'un ordinateur peut être amené à saisir des valeurs à travers un clavier (dans le cas de programmes interactifs). Pour permettre à l'ordinateur de lire les valeurs saisies par un utilisateur on va utiliser l'instruction `LIRE` ou `SAISIR` :

```
SAISIR <nom de variable>
```

ou

```
LIRE <nom de variable>
```

Cette instruction consiste à stocker en mémoire la valeur saisie par un utilisateur. Plus spécifiquement, la valeur est stockée dans la variable specifiée dans l'instruction.

4 Affichage à l'écran

Pour afficher des informations sur l'écran, on indique une liste d'expressions à afficher séparées par des virgules :

```
AFFICHER <expression>, .., <expression>
```

4.1 Exemple

L'instruction suivante permet d'afficher le texte indiqué à l'écran :

```
AFFICHER "Veuillez saisir votre nom:"
```

4.2 Exemple

L'instruction suivante permet d'afficher le contenu des variables x, y et z à l'écran :

```
AFFICHER x, y, z
```

4.3 Exemple

```
AFFICHER "Le résultat est: ", resultat
```

Cette instruction permet d'afficher un message suivi par la valeur de la variable `resultat`.

4.4 Exemple

```
AFFICHER "Veuillez saisir votre nom"
SAISIR nom
AFFICHER "Bonjour ", nom
```

5 Sélections

Ces instructions permettent de poser une condition à l'exécution d'un certain bloc d'instructions.

5.1 SI-ALORS

5.1.1 Syntaxe

```
SI <condition> ALORS
  <une ou plusieurs instructions>
FINSI
```

Tout d'abord, la condition se trouvant entre `SI` et `ALORS` est évaluée pour savoir si elle est *vraie* ou *fausse*.

Les instructions se trouvant entre `ALORS` et `FINSI` sont exécutées uniquement si la `condition` se trouvant entre `SI` et `ALORS` est vraie.

5.1.2 Exemple

```
AFFICHER "Veuillez saisir un entier"
SAISIR entier

SI entier < 0 ALORS
  AFFICHER "Valeur saisie strictement négative"
FINSI

SI entier > 0 ALORS
  AFFICHER "Valeur saisie strictement positive"
FINSI

AFFICHER "Fin de l'algorithme"
```

Voici un exemple d'exécution de cet algorithme :

```
Veuillez saisir un entier
5
La valeur saisie est strictement positive
Fin de l'algorithme
```

Voici un autre exemple d'exécution :

```
Veuillez saisir un entier
-7
La valeur saisie est strictement négative
Fin de l'algorithme
```

Voici un autre exemple d'exécution :

```
Veuillez saisir un entier
0
Fin de l'algorithme
```

5.2 SI-ALORS-SINON

5.2.1 Syntaxe

```
SI <condition> ALORS
  <une ou plusieurs instructions>
SINON
  <une ou plusieurs instructions>
FINSI
```

Tout d'abord, la condition se trouvant entre `SI` et `ALORS` est évaluée pour savoir si elle est *vraie* ou *fausse*.

Les instructions se trouvant entre `ALORS` et `SINON` sont exécutées uniquement si la `condition` se trouvant entre `SI` et `ALORS` est vraie. Dans ce cas, les instructions se trouvant entre `SINON` et `FINSI` ne sont pas exécutées.

Les instructions se trouvant entre `SINON` et `FISI` sont exécutées uniquement si la `condition` se trouvant entre `SI` et `ALORS` est fausse. Dans ce cas, les instructions se trouvant entre `ALORS` et `SINON` ne sont pas exécutées.

5.2.2 Exemple

```
AFFICHER "Veuillez saisir un entier"
SAISIR entier

SI entier < 0 ALORS
  AFFICHER "La valeur saisie est strictement négative"
SINON
  AFFICHER "La valeur saisie est positive ou nulle"
```

```
FINSI

AFFICHER "Fin de l'algorithme"
```

Voici un exemple d'exécution de cet algorithme :

```
Veuillez saisir un entier
5
La valeur saisie est positive ou nulle
Fin de l'algorithme
```

Voici un autre exemple d'exécution :

```
Veuillez saisir un entier
-8
La valeur saisie est strictement négative
Fin de l'algorithme
```

Voici un autre exemple d'exécution :

```
Veuillez saisir un entier
0
La valeur saisie est positive ou nulle
Fin de l'algorithme
```

5.3 SI-ALORS-SINONSI

On utilise cette structure pour effectuer plusieurs tests successifs.

5.3.1 Syntaxe

```
SI <condition> ALORS
  <une ou plusieurs instructions>
SINONSI <condition> ALORS
  <une ou plusieurs instructions>
SINON
  <une ou plusieurs instructions>
FINSI
```

5.3.2 Exemple

```
AFFICHER "Veuillez saisir un numero de mois"
SAISIR mois

SI mois = 1 ALORS
  AFFICHER "Janvier"
SINONSI mois = 2 ALORS
  AFFICHER "Février"
SINONSI mois = 3 ALORS
  AFFICHER "Mars"
SINONSI mois = 4 ALORS
  AFFICHER "Avril"
SINONSI mois = 5 ALORS
  AFFICHER "Mai"
SINONSI mois = 6 ALORS
  AFFICHER "Juin"
SINONSI mois = 7 ALORS
  AFFICHER "Juillet"
SINONSI mois = 8 ALORS
  AFFICHER "Août"
SINONSI mois = 9 ALORS
  AFFICHER "Septembre"
SINONSI mois = 10 ALORS
  AFFICHER "Octobre"
SINONSI mois = 11 ALORS
  AFFICHER "Novembre"
SINONSI mois = 12 ALORS
  AFFICHER "Décembre"
SINON
  AFFICHER "Mois inconnu"
FINSI

AFFICHER "Fin de l'algorithme"
```

Le Pseudo-code suivant est équivalent au précédent mais il occupe plus de lignes et nécessite plus de décalage vers la droite et il est moins lisible (l'algorithme précédent est donc préférable) :

```
AFFICHER "Veuillez saisir un numero de mois"
SAISIR mois

SI mois = 1 ALORS
  AFFICHER "Janvier"
SINON
  SI mois = 2 ALORS
    AFFICHER "Février"
  SINON
    SI mois = 3 ALORS
      AFFICHER "Mars"
    SINON
      SI mois = 4 ALORS
        AFFICHER "Avril"
      SINON
        SI mois = 5 ALORS
          AFFICHER "Mai"
        SINON
          SI mois = 6 ALORS
            AFFICHER "Juin"
          SINON
            SI mois = 7 ALORS
              AFFICHER "Juillet"
            SINON
              SI mois = 8 ALORS
                AFFICHER "Août"
              SINON
                SI mois = 9 ALORS
                  AFFICHER "Septembre"
                SINON
                  SI mois = 10 ALORS
                    AFFICHER "Octobre"
                  SINON
                    SI mois = 11 ALORS
                      AFFICHER "Novembre"
                    SINON
                      SI mois = 12 ALORS
                        AFFICHER "Décembre"
                      SINON
                        AFFICHER "Mois inconnu"
                      FINSI
                    FINSI
                  FINSI
                FINSI
              FINSI
            FINSI
          FINSI
        FINSI
      FINSI
    FINSI
  FINSI
FINSI

AFFICHER "Fin de l'algorithme"
```

6 Boucles

Les boucles sont des structures qui permettent de répéter des bouts de code un certain nombre de fois.

Dans ce cours, nous verrons les structures suivantes :

- `TANTQUE-FAIRE`
- `FAIRE-TANTQUE`
- `POUR`

6.1 TANTQUE-FAIRE

6.1.1 Syntaxe

```
TANTQUE <condition> FAIRE
  <une ou plusieurs instructions>
FINTANTQUE
```

Tout d'abord, la condition se trouvant entre `TANTQUE` et `FAIRE` est évaluée pour savoir si elle est *vraie* ou *fausse*.

Les instructions se trouvant entre `FAIRE` et `FINTANTQUE` sont exécutées tant que la `condition` se trouvant entre `TANTQUE` et `FAIRE` demeure vraie.

Notez que la condition est toujours évaluée avant le bloc d'instructions. Ainsi, il se peut que le bloc d'instructions ne soit jamais exécuté.

6.1.2 Exemple : Boucle de validation

On donne ici un exemple d'une *boucle de validation*.

```
AFFICHER "Veuillez saisir un mois"
SAISIR mois

TANTQUE mois < 1 OU mois > 12 FAIRE
  AFFICHER "Erreur: mois non valide"
  AFFICHER "Veuillez saisir un mois"
  SAISIR mois
FINTANTQUE

AFFICHER "Fin de l'algorithme"
```

Voici un exemple d'exécution de cet algorithme :

```
Veuillez saisir un mois
15
Erreur: mois non valide
Veuillez saisir un mois
0
Erreur: mois non valide
Veuillez saisir un mois
2
Fin de l'algorithme
```

Voici un autre exemple d'exécution :

```
Veuillez saisir un mois
4
Fin de l'algorithme
```

6.1.3 Exemple : Boucle avec compteur et accumulateur

```
compteur ← 0
accumulateur ← 0
AFFICHER "Veuillez saisir un entier positif ou nul"
SAISIR entier
accumulateur ← accumulateur + entier

TANTQUE entier < 0 FAIRE
  compteur ← compteur + 1
  AFFICHER "Erreur: entier non valide"
  AFFICHER "Veuillez saisir un entier positif ou nul"
  SAISIR entier
  accumulateur ← accumulateur + entier
FINTANTQUE

AFFICHER "Nombre d'erreurs: ", compteur
AFFICHER "Somme des entiers saisis: ", accumulateur
AFFICHER "Fin de l'algorithme"
```

Voici un exemple d'exécution de cet algorithme :

```
Veuillez saisir un entier positif ou nul
-15
Erreur: entier non valide
Veuillez saisir un entier positif ou nul
-5
Erreur: entier non valide
Veuillez saisir un entier positif ou nul
2
Nombre d'erreurs: 2
Somme des entiers saisis: -18
Fin de l'algorithme
```

6.1.4 Exemple : Boucle infinie

Il faut faire attention pour ne pas avoir une boucle sans fin. Une boucle sans fin est une boucle dont la condition est *toujours* vraie.

Voici un exemple :

```
compteur ← 0
accumulateur ← 0

TANTQUE compteur < 10 FAIRE
  accumulateur ← accumulateur + entier
  AFFICHER "Veuillez saisir un entier positif ou nul"
FINTANTQUE
```

Dans cet exemple, la boucle est sans fin car aucune instruction entre `FAIRE` et `FINTANTQUE` ne permet de changer la valeur de la variable `compteur` ce qui fait que la condition ne change jamais et reste vraie en tout temps.

Voici un autre exemple :

```
compteur ← 0
accumulateur ← 0

TANTQUE compteur < 10 FAIRE
  compteur ← compteur - 1
  accumulateur ← accumulateur + entier
  AFFICHER "Veuillez saisir un entier positif ou nul"
FINTANTQUE
```

Dans cet exemple, malgré que nous avons une instruction entre `FAIRE` et `FINTANTQUE` qui change la valeur de la variable `compteur`, ce changement de valeur ne permet pas d'atteindre une situation où la condition devient vraie.

6.2 FAIRE-TANTQUE

6.2.1 Syntaxe

```
FAIRE
  <une ou plusieurs instructions>
TANTQUE <condition>
```

Tout d'abord, Les instructions se trouvant entre `FAIRE` et `TANTQUE` sont exécutées. Ensuite, la condition se trouvant après `TANTQUE` est évaluée pour savoir si elle est *vraie* ou *fausse*. Tant que cette condition demeure vraie, les instructions se trouvant entre `FAIRE` et `TANTQUE` sont exécutées.

Notez que la condition est toujours évaluée après le bloc d'instructions. Ainsi, le bloc d'instructions est exécuté au moins une fois.

Cette instruction est similaire à la boucle `TANTQUE-FAIRE`, voici un exemple :

6.2.2 Exemple

```
compteur ← 0
accumulateur ← 0
FAIRE
  compteur ← compteur + 1
  accumulateur ← accumulateur + compteur
TANTQUE compteur < 10

AFFICHER "Somme des entiers de 1 à 10: ", accumulateur
```

On peut reproduire le même comportement à l'aide d'une boucle `TANTQUE-FAIRE` :

```
compteur ← 0
accumulateur ← 0
TANTQUE compteur < 10 FAIRE
  compteur ← compteur + 1
  accumulateur ← accumulateur + compteur
FINTANTQUE

AFFICHER "Somme des entiers de 1 à 10: ", accumulateur
```

Notez que dans cet exemple, les deux boucles utilisent le même nombre d'instructions.

6.2.3 Exemple : Boucle de validation

```
FAIRE
  AFFICHER "Veuillez saisir un mois"
  SAISIR mois
  SI mois < 1 OU mois > 12 ALORS
    AFFICHER "Erreur: mois non valide"
  FINSI
TANTQUE mois < 1 OU mois > 12

AFFICHER "Fin de l'algorithme"
```

Comparée à la boucle de validation illustrée avec la boucle `TANTQUE-FAIRE`, cette boucle permet d'éviter la duplication de code (les instructions d'affichage et de saisie au clavier) au coût d'une double évaluation de la condition d'arrêt de la boucle.

6.3 POUR-FAIRE

Cette boucle permet de répéter un bloc d'instructions un nombre fixe de fois.

6.3.1 Syntaxe

```
POUR <compteur> DE <valeur initiale> A <valeur finale> FAIRE
  <une ou plusieurs instructions>
FINPOUR
```

6.3.2 Exemple

```
AFFICHER "Veuillez saisir un chiffre"
SAISIR chiffre

POUR compteur DE 1 A 10 FAIRE
  resultat ← compteur × chiffre
  AFFICHER compteur, " multiplié par ", chiffre, " donne ", resultat
FINPOUR
```

7 Du Pseudo-code au Java

7.1 Affichage

Pseudo-code	`AFFICHER "Bonjour!"`
Java	`System.out.println( "Bonjour!" );`

7.2 Saisie

Pseudo-code	`SAISIR nomDeVariable`
	ou
	`LIRE nomDeVariable`

Java	`// on suppose ici que la variable est de type int` `nomDeVariable = Clavier.lireInt();`

7.3 Affectation

Pseudo-code	`nomDeVariable` $\leftarrow$ `5`
Java	`nomDeVariable = 5;`

7.4 SI-ALORS

Pseudo-code	`SI a = b ALORS`
	`  n ← 3`
	`  k ← 5`
	`FINSI`

Java	`if ( a == b ) {`
	`  n = 3;`
	`  k = 5;`
	`}`

En Java, dans le cas où un bloc contient une seule instruction on peut ne pas mettre d'accolades pour indiquer le début et la fin du bloc.

7.5 SI-ALORS-SINON

Pseudo-code	`SI a = b ALORS`
	`  n ← 3`
	`SINON`
	`  n ← 5`
	`FINSI`

Java	`if ( a == b ) {`
	`  n = 3;`
	`} else {`
	`  n = 5;`
	`}`

7.6 SI-ALORS-SINONSI

Pseudo-code	`SI a = b ALORS`
	`  n ← 3`
	`SINONSI a = 3 ALORS`
	`  n ← 5`
	`SINON`
	`  n ← 7`
	`FINSI`

Java	`if ( a == b ) {`
	`  n = 3;`
	`} else if ( a == 3 ) {`
	`  n = 5;`
	`} else {`
	`  n = 7;`
	`}`

7.7 TANTQUE-FAIRE

Pseudo-code	`TANTQUE a = b FIARE`
	`  a ← a + 1`
	`FINTANTQUE`

Java	`while ( a == b ) {`
	`  a = a + 1;`
	`}`

7.8 FAIRE-TANTQUE

Pseudo-code	`FAIRE`
	`  nomDeVariable ← nomDeVariable + 1`
	`TANTQUE nomDeVariable < 10`

Code Java	`do {`
	`  nomDeVariable = nomDeVariable + 1;`
	`} while ( nomDeVariable < 10 );`

7.9 POUR

Pseudo-code	`POUR i DE 1 À 10 FAIRE`
	`  AFFICHER i`
	`FINPOUR`

Code Java	`for ( int i = 1; i <= 10; i = i + 1 ) {`
	`  System.out.println( i );`
	`}`

8 Exercices

1. Soit l'algorithme suivant :

```
AFFICHER "Veuillez saisir un entier strictement positif"
SAISIR n

TANTQUE n ≤ 0 FAIRE
  AFFICHER "Erreur"
  AFFICHER "Veuillez saisir un entier strictement positif"
  SAISIR n
FINTANTQUE

AFFICHER "Correct"
```

   (a) De combien de tentatives dispose un utilisateur pour saisir une valeur strictement positive ?

   L'utilisateur dispose d'un nombre illimité de tentatives pour saisir une valeur positive.

   (b) Modifiez cet algorithme de façon à limiter le nombre de tentatives à un maximum de 10 tentatives.

```
AFFICHER "Veuillez saisir un entier strictement positif"
SAISIR n
nombreTentatives ← 1

TANTQUE n ≤ 0 ET nombreTentatives < 10 FAIRE
  AFFICHER "Erreur"
  AFFICHER "Veuillez saisir un entier strictement positif"
  SAISIR n
  nombreTentatives ← nombreTentatives + 1
FINTANTQUE

SI n > 0 ALORS
  AFFICHER "Correct"
FINSI
```

   (c) Écrivez votre algorithme en Java

```
System.out.println( "Veuillez saisir un entier strictement positif" );
n = Clavier.lireInt();
nombreDeTentatives = 1;

while ( n <= 0 && nombreDeTentatives < 10 ) {
    System.out.println( "Erreur" );
    System.out.println( "Veuillez saisir un entier strictement positif" );
    n = Clavier.lireInt();
    nombreDeTentatives = nombreDeTentatives + 1;
}

if ( n > 0 ) {
    System.out.println( "Correct" );
}
```

2. Soit l'algorithme suivant.

```
DEBUT
    AFFICHER "Saisir un entier:"
    SAISIR n
    SI n > 0 ALORS
      somme ← n × ( n + 1 ) / 2
    SINON
      somme ← 0
    FINSI
    AFFICHER "La somme est: ", somme
FIN
```

   On vous demande d'écrire un algorithme qui fait la même chose que l'algorithme précédent. Votre algorithme doit utiliser une seule instruction `SI-ALORS` de première forme (un `SI` sans le bloc `SINON`).

```
DEBUT
    AFFICHER "Saisir un entier:"
    SAISIR n
    somme ← 0
    SI n > 0 ALORS
      somme ← n × ( n + 1 ) / 2
    FINSI
    AFFICHER "La somme est: ", somme
FIN
```

3. L'algorithme suivant demande à l'utilisateur de saisir un nombre entier. Si l'utilisateur saisit un nombre strictement positif, par exemple la valeur 5, l'algorithme calcule la somme $1 + 2 + 3 + 4 + 5$ et afficher ensuite le résultat de ce calcul, soit le nombre 15.
   Si l'utilisateur saisit un nombre négatif ou nul, l'algorithme affiche 0.

```
DEBUT
  Afficher "Saisir un entier:"
  Lire n
  SI n > 0 ALORS
    somme ← n * ( n + 1 ) / 2
  SINON
    somme ← 0
  FIN SI
  Afficher "La somme est: ", somme
FIN
```

   On vous demande d'écrire un algorithme en Pseudocode qui fait la même chose que cet algorithme sans utiliser la structure `SI` et sans utiliser les opérations de divison et de multiplication.

Solution

```
DEBUT
  Afficher "Saisir un entier:"
  Lire entierSaisi

  somme ← 0
  entierCalcule ← entierSaisi

  TANT QUE entierCalcule > 0 FAIRE
    somme ← somme + entierCalcule
    entierCalcule ← entierCalcule - 1
  FIN TANT QUE

  Afficher "La somme est: ", somme
FIN
```

9 Références

—