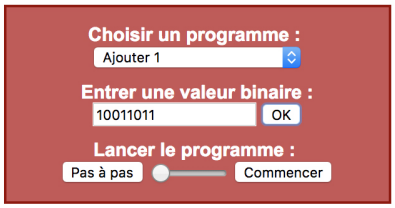

Etat	Lit	Ecrit	Déplace	Suivant
e1	VIDE	VIDE	gauche	e2
e2	0	0	gauche	e2
	1	1	gauche	e2
	VIDE	VIDE	droite	e3
e3	0	1	droite	fin
	1	0	droite	e3
	VIDE	1	droite	fin

Représentation algorithmique : le pseudo-code

Julien Noyer

2018-09-20

Le pseudo-code *Représentation d'un algorithme*

En algorithmie, le pseudo-code - ou LDA pour Langage de Description d'Algorithmes - est une façon de décrire un algorithme dans un langage presque naturel et limitant un maximum les références propres à tel ou tel langage de programmation : le but d'un algorithme est de pouvoir être traduit dans n'importe quel langage.

L'écriture en pseudo-code permet de prendre toute la mesure de la difficulté de la mise en œuvre de l'algorithme, et de développer une démarche structurée dans la construction de celui-ci. Son aspect descriptif permet de décrire en détail l'algorithme et permet une vision qui passe outre certains aspects complexes propre à la programmation informatique.

Dans la mesure ou il n'existe pas réellement de règles ou de convention établie pour l'écriture en pseudo-code, il est néanmoins nécessaire d'établir une structure - ou vocabulaire - qui puisse être comprit pas le plus grand nombre. Nous allons définir dans ce document la convention à suivre dans le cadre de la session de cours.

Structure d'un algorithme en pseudo-code

Dans la mesure ou notre pseudo-code doit représenter notre algorithme qui à pour but d'être développer et installer sur une machine, il faut respecter une structure informatique pour organiser le pseudo-code. Tous les pseudo-code doivent absolument respecter la structure suivante :

```
ALGORITHME <name_algo>
  // Déclarer les variables de l'algorithme

START
  // Effectuer les functions de l'algorithme

END
```

> Quel que soit votre algorithme, il doit toujours contenir cette structure.

Règles générales

Chaque mot-clefs sera écrit en majuscules :

- **FONCTION**
- **TANT QUE**
- **SI / FIN SI**
- **SELON**
- **POUR**

Le code peut être écrit en français - mais il est préférable d'utiliser l'anglais :

- FONCTION -> **FUNCTION**
- TANT QUE -> **WHILE**
- SI / FIN SI -> **IF / END IF**
- SELON -> **SWITCH**
- POUR -> **FOR**

Les noms des variables et des fonctions sont en ASCII en sans espaces :

- user_age
- user_name

Les commentaire sont écrit :

- // De cette manière
- /* De cette manière - multi-ligne */

Typer une variable

En algorithmie il est primordiale de typer les variables pour rendre la lecture du pseudo-code efficace. Les types possible utilisables sont :

- **INTEGER** : 14, -345
- **FLOAT** : 1.4, -76.98
- **BOOLEAN** : True/False
- **CHAR** : « a », « T »
- **STRING** : « Hello », « bonjour »
- **ARRAY** : [« algorithmie », true, 18]
- : USER, SKILLS

Pour déclarer une variable et la typer, il faut donner un nom à la variable et ajouter le type après le symbol **:** comme dans les exemple ci-dessous :

- user_age: **INTEGER**
- user_name: **STRING**

Pour affecter une valeur à une variables il faut utiliser une flèche droite/gauche de cette manière :

- user_age **<- 39**
- user_name **<- « Julien »**

> Il est possible de déclarer une variable, son type et son contenu d'une seule fois : `user_age: INTEGER <- 39`

Les **tableaux** sont des collections permettant de stocker des données afin d'effectuer des boucle sur ces données. Les tableaux peuvent être de type unique ou multiple :

```
// Déclaration
notes: ARRAY<Number>
options: ARRAY<STRING | BOOLEAN>

// Affectation
notes <- [12, 15, 9]
options <- [ true, "Off" ]
```

Un est une notion qui reprend la logique objet des langage de programmation. Selon notre programme nous pouvons avoir besoin de variables spécifiques que nous créerons de la manière suivante :

```
// Création
OBJECT User:
  name: STRING
```

```
  age: INTEGER

END OBJECT

// Affectation
newUser: User

// Assignation
newUser <- { "Juien", 39 }
```

> L'affectation et l'assignation peuvent être fait en une fois : `newUser: User <- { "Juien", 39 }`

Les conditions

En programmation une condition **SI / ET SI / SINON** permet de tester une ou plusieurs variables pour lancer une ou des fonctions spécifiques :

```
IF user_age < 12 THEN
  PRINT("User is a kid")

ELSE ID user_age < 18
  PRINT("User is a teenager")

ELSE
  PRINT("User is major")

END IF
```

Lorsque qu'une condition comprend plusieurs possibilités pré-établie, il faut utiliser la boucle **SELON** :

```
// Declaration
user_gender: INTEGER <- 2

SWITCH user_gender
  1 : PRINT("User is a woman")
  2 : PRINT("User is a man")
  3 : PRINT("User is a bot")
  DEFAULT PRINT("User gender unknow")
```

```
END SWITCH
```

Les boucles

Lorsque l'on traite de l'information en chaine il est souvent - voir toujours - utile de boucler sur une variable pour répéter un opération. Pour une boucle simple il faut utiliser la méthode **TANT QUE** :

```
// Declaration
i: INTEGER <- 20

WHILE i < 10
  PRINT("The value of 'i' is: " + i)
  i <- i + 1

END WHILE
```

Il est possible d'utiliser le boucle **TANT QUE** d'une manière alternative :

```
// Declaration
i: INTEGER <- 20

REPEATE
  PRINT("The value of 'i' is: " + i)
  i <- i + 1

WHILE i < 10
```

La méthode **POUR** permet également de faire des boucles mais avec une syntaxe différentes, plus proche des langage de programmation :

```
// Declaration
i: INTEGER <- 20

FOR i FROM 0 TO 10 [ i <- i + 1 ]
  PRINT("The value of 'i' is: " + i)

END FOR
```

> Il est conseiller de s'habituer à la syntaxe de la boucle **POUR**.

Les fonctions

Une **fonction** est un ensemble de d'instructions regroupées en une méthode qu'il est possible de déclencher à tous moment dans notre pseudo-code - dans la mesure ou la déclaration précède l'exécution. Une fonction et un petit algorithme à elle toute seule, il est donc possible d'y déclarer des variables mais elles ne seront disponible hors de la fonction que si elle sont retourner - avec le mot clef **RETURN**.

```
FUNCTION add(a: INTEGER, b: INTEGER)
  result: INTEGER

  START
    result <- a + b
    RETURN result

  END
/FUNCTION
```