

Développement Web (R112)


MMI, IUT1, UGA

Gwen Salaün

Algorithmique

(cours 3)

Just like reading and writing...



“In fifteen years
we’ll be teaching
programming just
like reading and
writing . . . and
wondering why we
didn’t do it sooner.”

— Mark Zuckerberg

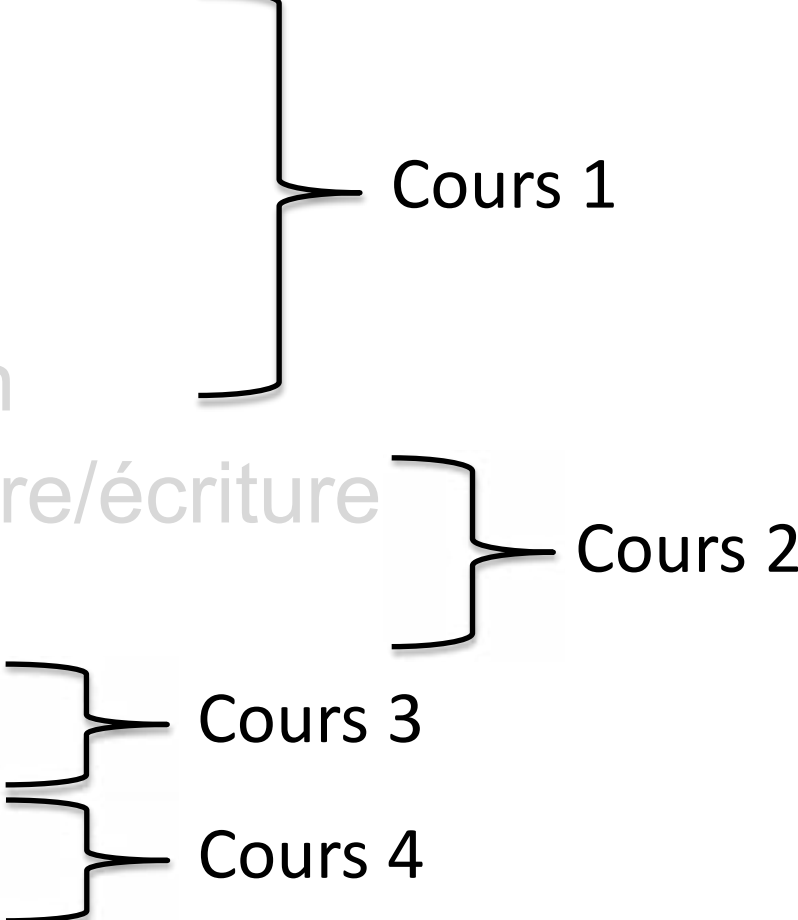
Try an Hour of Code for
Computer Science Education Week
December 9–15.
Anybody can learn!

<http://code.org>

Computer
Science
Education
Week
December 9–15, 2014

C O
D E

Plan

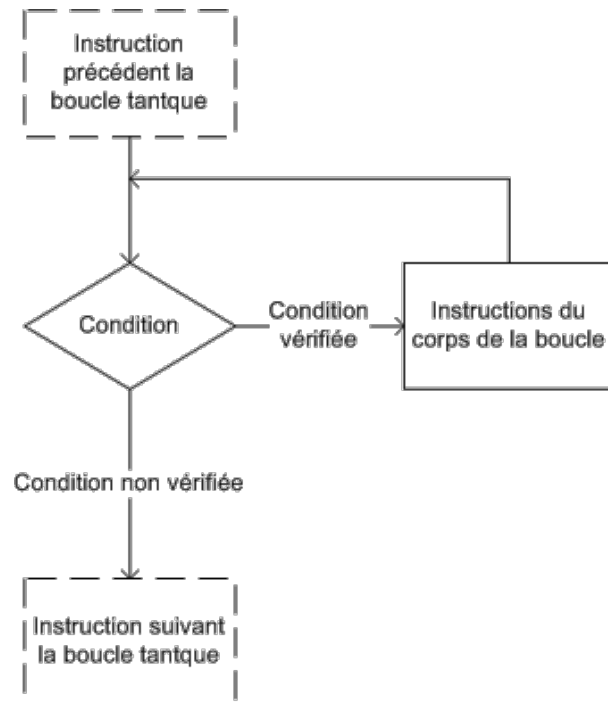
- Programmation
 - Algorithmique
 - Notion de variable
 - L'instruction d'affectation
 - Les instructions de lecture/écriture
 - Le choix
 - Les boucles
 - Factorisation du code
- 
- Cours 1
- Cours 2
- Cours 3
- Cours 4

Choix vs boucle

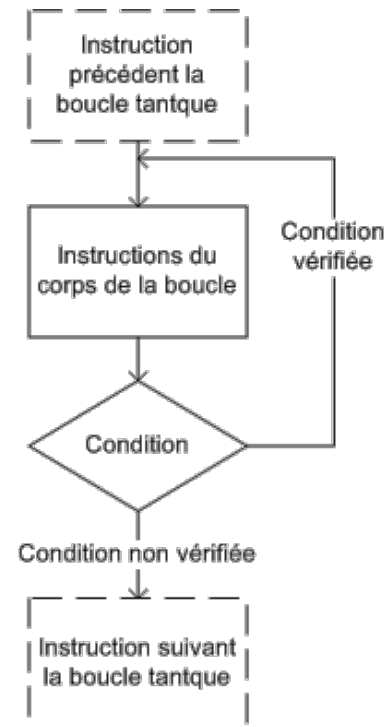
- Nous pouvons en fonction de conditions exécuter telle ou telle séquence d'instructions (choix/test 'si .. alors .. sinon .. fin si')
- Mais il arrive qu'un même traitement se répète plusieurs fois avec des valeurs possiblement différentes pour les variables
- Solution : utiliser des boucles nommées **structures répétitives** ou **itératives**

Boucles 'tant que'

Tant que (condition) **Faire**
instructions
Fin Tant que



Faire
instructions
Tant que (condition)



Boucle 'tant que' – exemple

Le client doit choisir de prendre un café ou non

Algo boucle1

var c : caractère

Début

Ecrire("Voulez vous un café ? (O/N)")

c ← Lire()

Tant que (c ≠ 'O') et (c ≠ 'N') **Faire**

c ← Lire()

Fin Tant que

Fin

Boucle infinie

Lorsqu'on oublie de modifier la condition de boucle de sorte qu'une fois entré dans une boucle, on ne puisse jamais en sortir

Algo boucle2

var i : entier

Début

i \leftarrow 0

Tant que (i<10) **Faire**

Ecrire(i)

Fin Tant que

Fin

Boucle inutile

Il faut vérifier que l'on puisse passer dans la boucle => Vérifier l'initialisation des variables de la condition

Algo boucle3

var i : entier

Début

i ← 11

Tant que (i<10) **Faire**

Ecrire(i)

Fin Tant que

Fin

Boucle – exemple (1)

- Écrire un algorithme qui demande successivement 20 nombres à l'utilisateur, et qui lui indique à la fin le plus grand parmi ces 20 nombres.
- Quel type de boucle choisir ?
 - Tant que .. → pas de passage obligatoire dans la boucle
 - Faire .. Tant que → au moins un passage dans la boucle
- Que faire dans la boucle ?
- Quelle est la condition de continuation de boucle ?

Solution avec 'faire'

Algo boucle4

```
var i : entier      // variable pour compter les nombres
var max : entier    // stockage du max
var n : entier      // dernière valeur lue en entrée
```

Début

```
i ← 1
```

```
max ← Lire()
```

Faire

```
  n ← Lire()
```

```
  Si (n > max) Alors
```

```
    max ← n
```

```
  Fin Si
```

```
  i ← i + 1
```

```
Tant que (i < 20)
```

```
  Ecrire ("Le maximum est " + max)
```

Fin

Solution avec 'tant que'

Algo boucle5

```
var i : entier      // variable pour compter les nombres
var max : entier    // stockage du max
var n : entier      // dernière valeur lue en entrée
```

Début

```
i ← 1
```

```
max ← Lire()
```

```
Tant que (i < 20) faire
```

```
    n ← Lire()
```

```
    Si (n > max) Alors
```

```
        max ← n
```

```
    Fin Si
```

```
    i ← i + 1
```

```
Fin Tant que
```

```
Ecrire ("Le maximum est " + max)
```

Fin

Boucle – exemple (2)

- Écrire un algorithme qui demande un nombre de départ, et qui calcule la somme des entiers jusqu'à ce nombre
- Par exemple, si l'on entre 5, le programme doit calculer :

$$1 + 2 + 3 + 4 + 5 = 15$$

- NB : on souhaite afficher uniquement le résultat, pas la décomposition du calcul

Solution avec 'faire'

Algo boucle7

```
var i : entier      // variable pour compter les entiers
var som : entier    // résultat de la somme
var n : entier      // valeur entrée par l'utilisateur
```

Début

```
i ← 0
som ← 0
n ← Lire()
```

Faire

```
    som ← som + i
    i ← i + 1
```

Tant que ($i \leq n$)

```
Ecrire ("La somme est " + som)
```

Fin

Pour $n=3$ quelles sont les valeurs de 'som' et 'i' à la fin?

Solution avec 'faire'

Algo boucle7

```
var i : entier      // variable pour compter les entiers
var som : entier    // résultat de la somme
var n : entier      // valeur entrée par l'utilisateur
```

Début

```
i ← 0
som ← 0
n ← Lire()
```

Faire

```
    som ← som + i
    i ← i + 1
```

Tant que ($i \leq n$)

Ecrire ("La somme est " + som)

Fin

Pour $n=3$ quelles sont les valeurs de 'som' et 'i' à la fin? → som=6 et i=4

Solution avec 'faire'

Algo boucle7

```
var i : entier      // variable pour compter les entiers
var som : entier    // résultat de la somme
var n : entier      // valeur entrée par l'utilisateur
```

Début

```
i ← 0
som ← 0
n ← Lire()
```

Faire

```
    som ← som + i
    i ← i + 1
```

Tant que ($i \leq n$)

```
    Ecrire ("La somme est " + som)
```

Fin

La condition de boucle est cruciale. Si on met $i \neq n$ on obtient alors $som=3$ et $i=3$ si on rentre n qui vaut 3 !

Pour $n=3$ quelles sont les valeurs de 'som' et 'i' à la fin? ➔ $som=6$ et $i=4$

Solution avec 'tant que'

Algo boucle8

```
var i : entier      // variable pour compter les entiers
var som : entier    // résultat de la somme
var n : entier      // valeur entrée par l'utilisateur
```

Début

```
i ← 0
som ← 0
n ← Lire()
```

Tant que ($i \leq n$) **faire**

```
    som ← som + i
    i ← i + 1
```

Fin Tant que

```
Ecrire ("La somme est " + som)
```

Fin

Pour $n=3$ quelles sont les valeurs de 'som' et 'i' à la fin?

Solution avec 'tant que'

Algo boucle8

```
var i : entier      // variable pour compter les entiers
var som : entier    // résultat de la somme
var n : entier      // valeur entrée par l'utilisateur
```

Début

```
i ← 0
som ← 0
n ← Lire()
```

Tant que ($i \leq n$) **faire**

```
    som ← som + i
    i ← i + 1
```

Fin Tant que

```
Ecrire ("La somme est " + som)
```

Fin

Pour $n=3$ quelles sont les valeurs de 'som' et 'i' à la fin? ➔ som=6 et i=4

Solutions alternatives

Algo boucle9

var som : entier

var n : entier

Début

som \leftarrow 0

n \leftarrow Lire()

Faire

som \leftarrow som + n

n \leftarrow n - 1

Tant que (n \neq 0)

Ecrire ("La somme est " +som)

Fin

Algo boucle10

var som : entier

var n : entier

Début

som \leftarrow 0

n \leftarrow Lire()

Tant que (n > 0) **faire**

som \leftarrow som + n

n \leftarrow n - 1

Fin Tant que

Ecrire ("La somme est " +som)

Fin

Pour n=3 quelle est la valeur de 'som' à la fin?

Solutions alternatives

Algo boucle9

var som : entier

var n : entier

Début

som \leftarrow 0

n \leftarrow Lire()

Faire

som \leftarrow som + n

n \leftarrow n - 1

Tant que (n \neq 0)

Ecrire ("La somme est " +som)

Fin

Algo boucle10

var som : entier

var n : entier

Début

som \leftarrow 0

n \leftarrow Lire()

Tant que (n > 0) **faire**

som \leftarrow som + n

n \leftarrow n - 1

Fin Tant que

Ecrire ("La somme est " +som)

Fin

Pour n=3 quelle est la valeur de 'som' à la fin? \rightarrow som=6

Boucle 'pour'

- La boucle 'pour' peut être utilisée lorsque l'on connaît a priori le nombre d'itérations
- La boucle 'pour' permet de faire varier un indice entre deux valeurs
- En pseudo-code :

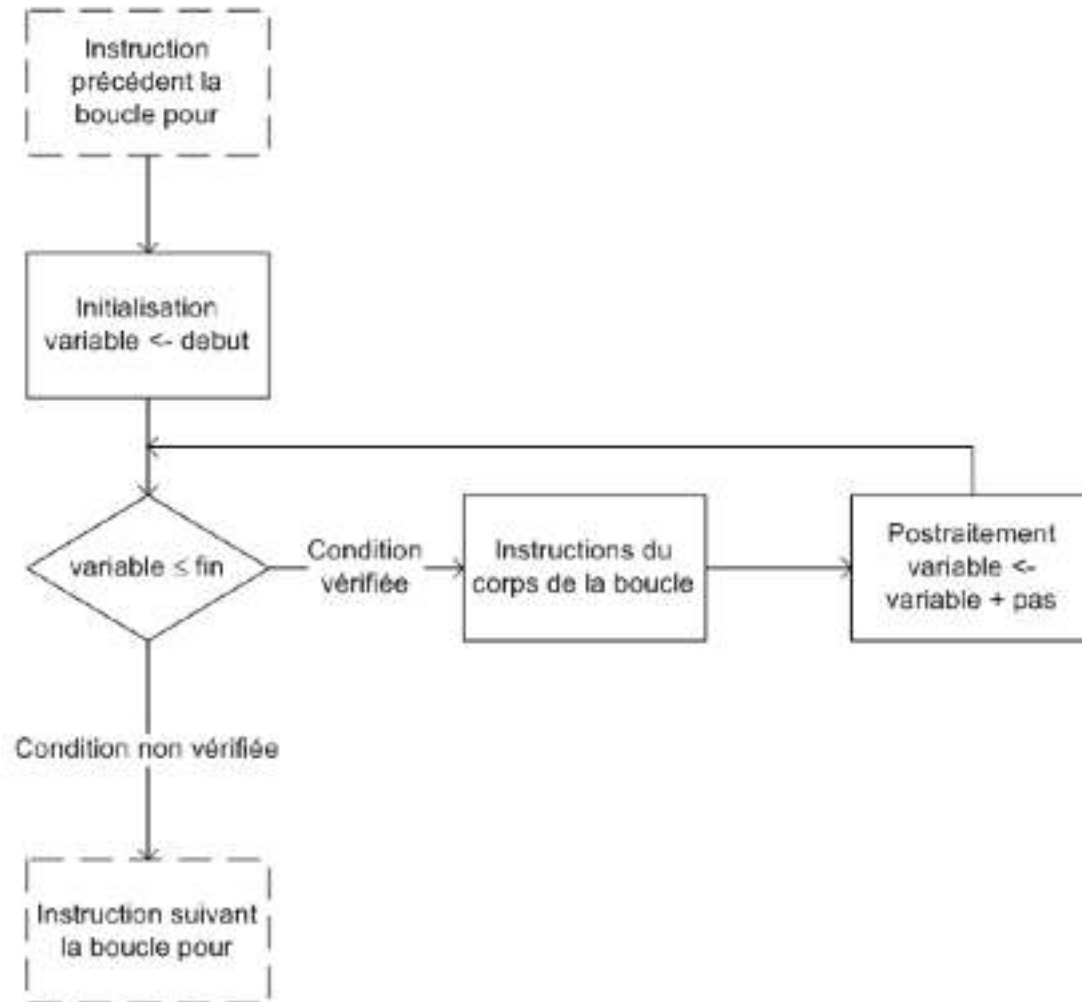
Pour variable **de** debut à fin **pas** de n

Faire

Instructions // la variable varie de debut a fin par pas de n

Fin Pour

Boucle 'pour'



Équivalence 'pour' et 'tant que'

Pour vari **de** debut **à** fin

Faire

instructions

Fin Pour



vari \leftarrow debut

Tant que (vari \leq fin) **faire**

Instructions

vari \leftarrow vari + 1

Fin Tant que

Pour i **de** 1 **à** 5 **pas** de 1

Faire

Ecrire(i)

Fin Pour



i \leftarrow 1

Tant que (i \leq 5) **faire**

Ecrire(i)

i \leftarrow i + 1

Fin Tant que

Boucle 'pour' – Compteur

Il ne faut pas modifier le compteur dans la boucle 'pour' !

Pour i de 1 à n pas 1

Faire

Ecrire(i)

$i \leftarrow i * 2$

Fin Pour



L'incrémentation
du compteur est
implicite

Boucles imbriquées

- Les boucles peuvent être contenues les unes dans les autres
- On parle de **boucles imbriquées**
- Exemple : afficher tous les couples possibles étant donné deux ensembles d'entiers, par exemple $\{1,2,3,4\}$ et $\{1,2,3,4,5\}$

Boucles imbriquées – exemple

Algo boucle11

var i: entier

var j : entier

Début

Ecrire(" {");

Pour i de 1 à 4 pas 1 faire

Pour j de 1 à 5 pas 1 faire

Ecrire ("(" + i + ", " + j + ")");

Fin pour

Fin pour

Ecrire ("}");

Fin

Ce qui donne { (1,1), (1,2), (1,3), (1,4), (1,5), (2,1), (2,2), ..²⁶ }

Au prochain cours

- La factorisation de code avec les fonctions