

Introduction à l'algorithmique

— La pensée logique du développement

Au programme

- Introduction
 - ❖ Atelier
- Chapitre 1: Logique conditionnelle
 - ✓ Les variables et les constantes
 - ✓ Les conditions: principe général
 - ✓ Opérateurs conditionnels
 - ✓ Opérateurs logiques
 - ✓ Structures conditionnelles
 - ❖ Atelier
- Chapitre 2: Logique itérative
 - ✓ Itération et répétitions: principe général
 - ✓ Les boucles
 - ❖ Atelier

Au programme

- Chapitre 3: Fonctions et récursivité
 - ✓ Les fonctions: l'art de l'algorithmique générique
 - ❖ Atelier
 - ✓ La récursivité

Atelier

Effectuez l'atelier n°9 sur Community
"Introduction à l'algorithmique"



Logique conditionnelle



Chapitre 1

Avertissement

⚠ Dans ce cours, nous allons pratiquer les notions de bases de l'algorithmique avec du **pseudo-code en français**. Il ne s'agit pas d'un vrai langage de programmation mais simplement d'une manière d'écrire des opérations logiques.



Les variables et les constantes

- ✓ Une **variable** est un nom (ou suite de caractères) auquel est associé une **valeur**. Son contenu est modifiable après sa déclaration.
 - ✓ Une **constante** est identique à une variable, mais sa valeur est **non-modifiable** après sa déclaration.
- ⚠ On peut **déclarer une variable** (dire qu'elle existe) **sans lui assigner immédiatement une valeur**. En revanche, une **constante** doit **avoir une valeur** lors de sa déclaration.

```
VARIABLE age = 30
```

```
AFFICHER age
```

```
age = 31
```

```
AFFICHER age
```

```
> 30
```

```
> 31
```

Exemple de variable

```
CONSTANTE taille = 180
```

```
AFFICHER taille
```

```
taille = 185
```

```
AFFICHER taille
```

```
> 180
```

```
> 180
```

Exemple de constante



Les conditions: principe général



Paramètres

Par exemple: saisies par l'utilisateur, mesures, données de sources externes

```
Instruction 1  
  
SI condition {  
    Instruction 2  
}  
SINON {  
    Instruction 3  
}
```



Résultat

Par exemple: Affichage sur un écran, génération de fichiers, impression, etc...

Un programme est un ensemble d'instructions acceptant (en général) des **paramètres** d'entrée et produisant un **résultat**.

Les **conditions** sont un moyen d'anticiper différentes situations. Autrement dit, écrire un programme revient à gérer un certain nombre de **possibilités** et à produire un résultat correspondant.



Opérateurs de comparaison

- ✓ Ecrire **une condition** revient, en règle générale, à **poser une question** (comparaison) dont **la réponse est vrai ou faux**. C'est pour cela que l'on retrouve dans tous les langages de programmation un ensemble **d'opérateurs de comparaison**.
- ✓ Opérateurs de comparaison communs et leurs notations communes:

Inférieur à	Supérieur à	Inférieur ou égal à	Supérieur ou égal à	Egal à	Différent de
<	>	<=	>=	==	!=

```
SI temperature <= 0 {  
    AFFICHER "Il fait froid"  
}
```

Exemple de condition



Opérateurs logiques

- ✓ Il est fréquent de devoir combiner plusieurs comparaison pour écrire les conditions plus complexes.
- ✓ Pour cela, on utilise les opérateurs logiques, permettant de combiner des comparaisons:

Et	Ou	Priorité	Négation
&& and	 or	()	! not

```
SI temperature <= 0 && humidité > 80 {  
    AFFICHER "Attention verglas"  
}
```

Exemple de condition



Opérateurs logiques

```
VARIABLE temperature = 25
VARIABLE meteo = "Soleil"

SI temperature > 15 && meteo == "Soleil" {
    AFFICHER "Il fait beau."
}
SINON {
    AFFICHER "Il ne fait pas beau."
}
```

> Il fait beau.

Dans l'exemple ci-contre, la comparaison doit être lue de la manière suivante:

`temperature > 20 && meteo == "Soleil"`

└──────────┘ └──────────┘

Température est supérieur à 20 ? Météo est égal à "Soleil" ?

Vrai ET Vrai

└──────────┘

Vrai



Structures conditionnelles

```
SI temperature > 15 && meteo == "Soleil" {  
    AFFICHER "Il fait beau."  
}  
SINON SI meteo == "Pluie" {  
    AFFICHER "Il pleut."  
}  
SINON {  
    AFFICHER "Il ne fait pas beau."  
}
```

La plupart des langages partagent une **structure conditionnelle** commune basée sur 3 mots clefs:

- ✓ **SI** *condition(s)* { ... }
- ✓ **SINON SI** *condition(s)* { ... }
- ✓ **SINON** { ... }

💡 On peut utiliser un **SI** tout seul ou enchaîner plusieurs **SINON SI**. En revanche un **SINON** arrive toujours en dernière position est n'a jamais de condition.



Atelier

Effectuez l'atelier n°10 sur Community
"Algorithme Resto'Commande - Partie 1"



Logique itérative



Chapitre 2

Itération et répétitions: principe général

- ✓ Lors de l'écriture d'un algorithme, il est fréquent de devoir **répéter certaines instructions**. Cette répétition peut avoir lieu un **nombre de fois déterminé** ou **indéterminé**.
- ✓ Une **boucle** permet la **répétition** d'une même série d'instructions selon une **condition**, on parle d'**itération** pour décrire cette répétition.

```
VARIABLE age = 30  
AFFICHER age  
age = 31  
AFFICHER age  
age = 32  
AFFICHER age  
age = 33  
AFFICHER age
```

> 30
> 31
> 32
> 33

Exemple de variable

```
VARIABLE age = 30  
  
TANT QUE age <= 33 {  
    AFFICHER age  
    age = age + 1  
}
```

> 30
> 31
> 32
> 33

Exemple de constante



Les boucles

```
VARIABLE reponse = "Non"
TANT QUE reponse != "Oui" {
    AFFICHER "On est arrivé?"
    reponse = LIRE_SAISIE_CLAVIER
}
AFFICHER "Trop bien."
```

```
> On est arrivé ?
SAISIE: Non
> On est arrivé ?
SAISIE: Non
> On est arrivé ?
SAISIE: Oui
> Trop bien.
```

La boucle **TANT QUE** est une boucle à nombre d'itérations indéterminé. Elle continue d'exécuter les instructions qu'elle contient **tant que sa condition est vraie**.

💡 En règle générale, la boucle **TANT QUE** est utilisée quand la condition dépend d'une **valeur extérieure au programme** et donc imprévisible comme une saisie de l'utilisateur.



Les boucles

```
POUR i = 30 TANT QUE i < 33 EFFECTUER i++) {  
    AFFICHER age  
}
```

```
> 30  
> 31  
> 32  
> 33
```

La boucle **POUR** définit un nombre d'itérations.

💡 La boucle **POUR** s'utilise dès lors que le nombre d'itérations dépend d'un compteur permettant de définir le nombre d'itérations.



Atelier

Effectuez l'atelier n°10 sur Community
"Algorithme Resto'Commande - Partie 2"



Fonctions et récursivité

Chapitre 3

Les fonctions: l'art de l'algorithmique générique


```
FONCTION verifierAge (age) {  
    SI age >= 18 {  
        RETOURNER "majeur"  
    }  
    Sinon {  
        RETOURNER 'enfant'  
    }  
  
    AFFICHER verifierAge(30)
```

> majeur

Une **fonction**, également appelée **sous-programme** en algorithmique, est un **groupe d'instructions** acceptant des **paramètres** et retournant un **résultat**. C'est une sorte de programme dans le programme.

💡 Une **fonction** permet de réutiliser un morceau de code sans avoir à le réécrire.

```
FONCTION verifierAge (age) { ... }
```


Nom de la fonction Paramètres (dans l'ordre)



Atelier

Effectuez l'atelier n°10 sur Community
"Algorithme Resto'Commande - Partie 3"



La récursivité

```
FONCTION factorielle (n) {  
    SI n == 0 {  
        Retourner 1  
    }  
    Sinon {  
        Retourner n * factorielle(n-1)  
    }  
}  
AFFICHER factorielle(3)
```

> 6

On nomme **récursive** une fonction s'appelant elle-même. Un tel procédé permet la résolution de situation nécessitant l'exécution successive d'une même tâche plusieurs fois.

💡 La **récursivité** est une **alternative** aux **boucles**. On l'utilise notamment pour effectuer des opérations comme le tri.

💡 Factorielle de 3 ou $!3 = 3 * 2 * 1$



La récursivité

```
FONCTION factorielle (n) {  
    SI n == 1 {  
        Retourner 1  
    }  
    Sinon {  
        Retourner n * factorielle(n-1)  
    }  
}  
AFFICHER factorielle(3)
```

> 6

L'exécution de l'exemple ci-contre peut se représenter de la manière suivante:

factorielle(3) =

3 * factorielle(2) =

3 * 2 * factorielle(1) =

3 * 2 * 1



```
17 string sInput;  
18 int iLength, iN;  
19 double dblTemp;  
20 bool again = true;
```

```
21 while (again) {  
22     iN = -1;  
23     again = false;  
24     getline(cin, sInput);  
25     system("cls");  
26     stringstream(sInput) >> dblTemp;  
27     iLength = sInput.length();  
28     if (iLength < 4) {  
29         again = true;  
30         continue;  
31     } else if (sInput[iLength - 3] != '.') {  
32         again = true;  
33         continue;  
34     } while (++iN < iLength) {  
35         if (isdigit(sInput[iN])) {  
36             continue;  
37         } else if (iN == (iLength - 1)) {  
38             continue;  
39         }  
40     }  
41     // ...  
42 }
```

Fin du cours