

FORMATION ALGORITHMIQUE

FRÉDÉRIC GIROD
ELORRI



INTRODUCTION

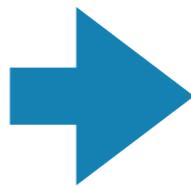
PROGRAMME

- Un programme correspond à une méthode de résolution pour un problème donné.
- Cette méthode de résolution est effectuée par une suite d'instructions d'un langage de programmation.
- Ces instructions permettent de traiter et de transformer les données (entrées) du problème à résoudre pour aboutir à des résultats (sorties).
- Un programme n'est pas une solution en soi mais une méthode à suivre pour trouver des solutions.

PROGRAMMATION

- Ensemble des activités orientées vers la conception, la réalisation, le test et la maintenance de programmes

Ce que vous pensez ?



Mon but !



LANGAGES INFORMATIQUES

- Le langage informatique est l'intermédiaire entre le programmeur et la machine.
- Il permet d'écrire des programmes destinés à effectuer une tache donnée.

LANGAGES DE PROGRAMMATION

- Il existe beaucoup de langages de programmation :
 - ➔ C++, Java, C#, PHP, JavaScript, Python, Swift, ...
- Le choix d'un langage de programmation n'est pas facile, chacun a ses spécificités et correspond mieux à certains types d'utilisations et parfois même à certains domaines.
- Chaque langage aura sa syntaxe bien particulière avec parfois des similitudes.

ALGORITHME

- Un algorithme est une suite d'instructions ayant pour but de résoudre un problème donné
- Un algorithme peut se comparer à une recette de cuisine
- Exemples d'algorithme que nous utilisons dans la vie :
 - ➔ Préparer une recette de cuisine
 - ➔ Montrer le chemin à un touriste
 - ➔ Monter un meuble en kit...

ALGORITHME

29 min (26,9 km)



via A25

Le plus rapide, malgré un trafic ralenti

- ▼ Prendre Rue du Printemps en direction de Rue Denis Papin

2 min (750 m)

- ▲ Rejoindre Rue Jean Jaurès/D14

130 m

- 📍 Au rond-point, prendre la 5e sortie sur Rue du Printemps

450 m

- ➡ Prendre à droite sur Rue Papin

110 m

- ➡ Tourner à droite au 1er croisement et continuer sur Rue Denis Papin

⚠ Voie à accès limité

ℹ Votre destination se trouvera sur la gauche.

78 m

EXAMPLE

Urbilog (SAS)

31 Rue Denis Papin, 59650 Villeneuve-d'Ascq

ALGORITHME ET PROGRAMMATION

- L'élaboration d'un algorithme précède l'étape de programmation
 - ➡ Un programme est un algorithme
 - ➡ Un langage de programmation est un langage compris par l'ordinateur
- La rédaction d'un algorithme est un exercice de réflexion

ALGORITHME ET PROGRAMMATION

- La rédaction d'un algorithme est un exercice de réflexion
 - ➔ L'algorithme est indépendant du langage de programmation
 - ➔ Par exemple, on utilisera le même algorithme pour une implantation en Java, en C# ou en PHP...
 - ➔ L'algorithme est la résolution brute d'un problème informatique

ALGORITHMIQUE

- L'algorithmatique désigne la discipline qui étudie les algorithmes et leurs applications en Informatique.
- Une bonne connaissance de l'algorithmatique permet d'écrire des algorithmes efficaces.
- Permet le développement de l'esprit logique.

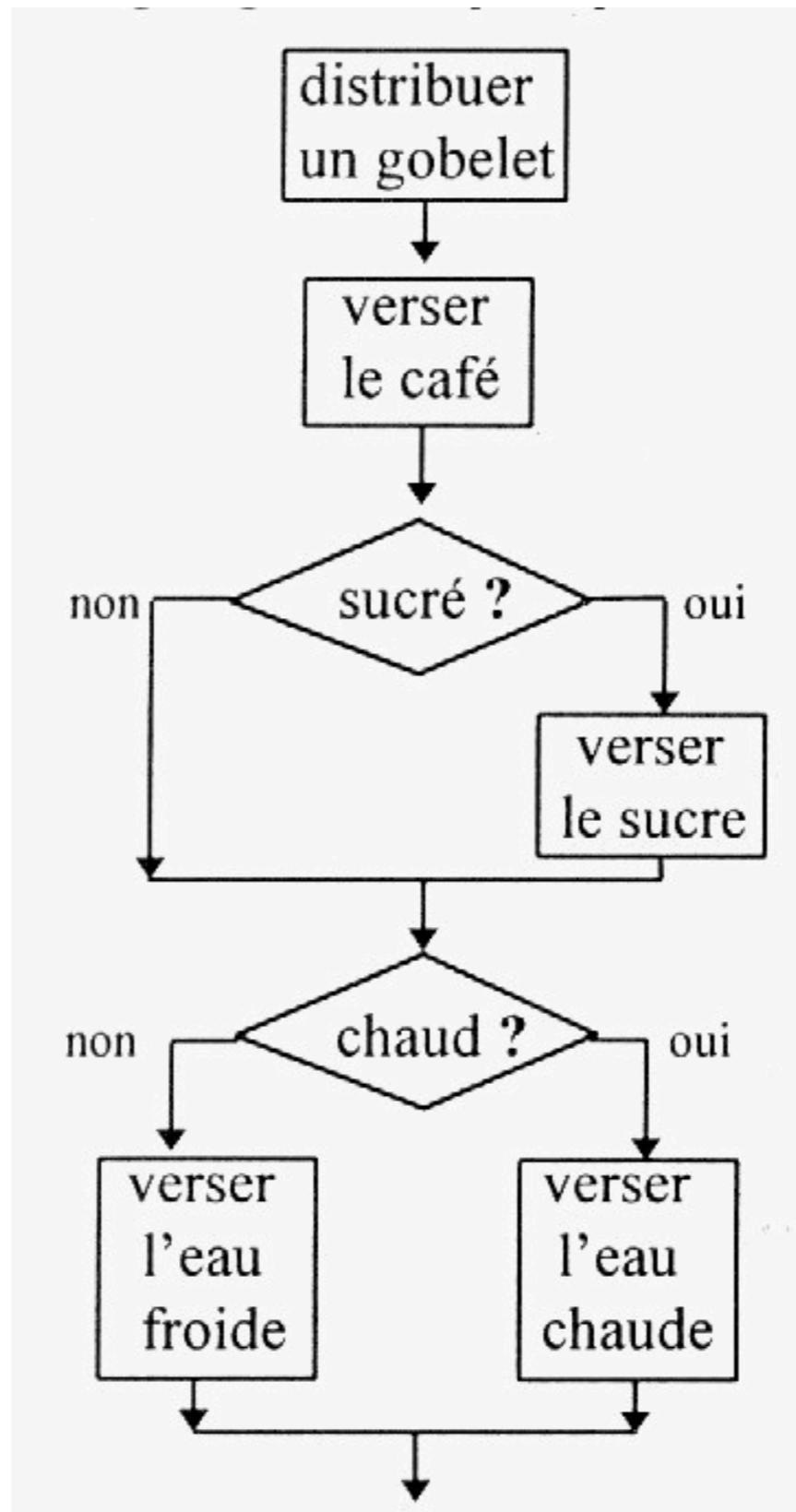
MÉMO
ALGORITHME = MÉTHODE DE RÉSOLUTION

REPRÉSENTATION D'UN ALGORITHME

- Deux façons de représenter un algorithme :
 - ➡ L'organigramme : représentation graphique avec des symboles (carrés, losanges, etc...)
 - ➡ Le pseudo-code : représentation textuelle avec une série de conventions ressemblant à un langage de programmation

REPRÉSENTATION D'UN ALGORITHME

EXEMPLE ORGANIGRAMME



REPRÉSENTATION D'UN ALGORITHME

EXEMPLE PSEUDOCODE

Algo distributeurCafé

Variables

sucre, chaud: booléen

Début

Ecrire("Voulez-vous du sucre ?")

Lire(sucre)

Ecrire("Boisson chaude ?")

Lire(chaud)

Si (sucre) Alors

 Ecrire("Ajout du sucre")

FinSi

Si (chaud) Alors

 Ecrire("Ajout de l'eau chaude")

Sinon

 Ecrire("Ajout de l'eau froide")

FinSi

Fin

REPRÉSENTATION D'UN ALGORITHME

- Deux façons de représenter un algorithme :
 - ➔ L'organigramme : représentation graphique avec des symboles (carrés, losanges, etc...)
 - ➔ Le pseudo-code : représentation textuelle avec une série de conventions ressemblant à un langage de programmation

NOTIONS ET INSTRUCTIONS INCONTOURNABLES EN ALGORITHMIQUE

INSTRUCTIONS DE BASE

- Un programme informatique est formé de quatre types d'instructions considérées comme la « base » de la programmation :
 - ➔ l'affectation de variables
 - ➔ la lecture et/ou l'écriture
 - ➔ les conditions
 - ➔ les boucles

POUR ALLER PLUS LOIN

POUR ALLER PLUS LOIN

- Pour aller plus loin les notions suivantes vont très vite devenir incontournables :
 - ➔ les fonctions
 - ➔ les tableaux numériques
 - ➔ les enregistrements ou tableaux associatifs ou objets

NOTIONS ET INSTRUCTIONS INCONTOURNABLES EN ALGORITHMIQUE

NOTION DE VARIABLE

- Une variable sert à stocker la valeur d'une donnée dans un langage de programmation
- Une variable désigne un emplacement dont le contenu peut changer au cours d'un programme (d'où le nom de variable)

NOTION DE VARIABLE

- Une variable se caractérise par :
 - ➔ un nom (ou identifiant)
 - ➔ un type (indique le type de valeur que peut prendre la variable, exemple : un nombre ou une chaîne de caractères)
 - ➔ une valeur
- Attention : la variable doit être déclarée avant d'être utilisée.

NOTION DE VARIABLE

- Le choix du nom d'une variable est soumis à quelques règles qui varient selon le langage, mais en général :
 - ➔ doit commencer par une lettre alphabétique
 - ➔ doit être constitué uniquement de lettres, de chiffres et du « tiret du bas » (_), pas d'espaces et de caractères de ponctuation

NOTION DE VARIABLE

- Conseil : pour la lisibilité et la compréhension du code, il est recommandé de choisir des noms significatifs qui décrivent les données manipulées
 - ➔ Exemples : NomRecette, Prix_TTC, etc...

TYPES DES VARIABLES

- Le type d'une variable détermine l'ensemble des valeurs qu'elle peut prendre.
- Les types proposés par la plupart des langages sont :
 - ➔ Type numérique (entier et flottant)
 - ➔ Type booléen
 - ➔ Type caractère
 - ➔ Type chaîne de caractère

DÉCLARATION DES VARIABLES

- En algorithmique, la déclaration de variables est effectuée par la forme suivante :
 - Variable maVariable : type
- Exemple : **variables**
 - nbMax : entier
 - porteOuverte : booléen
 - prenom, nom : chaîne de caractères

AFFECTATION D'UNE VARIABLE

- L'affectation consiste à attribuer une valeur à une variable.
- En algorithmique, l'affectation est notée par le signe <-
- Exemple : maVariable <- 12
- L'affectation ne modifie que ce qui est à gauche de la flèche.

AFFECTATION D'UNE VARIABLE

- La plupart des langages de programmation utilisent le signe égal = pour l'affectation <–
- Lors d'une affectation, l'expression de droite est évaluée et la valeur trouvée est affectée à la variable de gauche
- Exemple : A <– B différent de B <– A

NOTION DE CONSTANTE

- Une constante est une variable dont la valeur ne change pas au cours de l'exécution du programme.
- En algorithmique, la déclaration d'une constante est effectuée par la forme suivante :
 - ➡ Constante maConstante<-valeur : type
- Exemple : Constantes
pi<-3.14 : réel
nombreMax<-128 : entier

SYNTAXE GÉNÉRALE D'UN ALGORITHME

```
Algo monPremierAlgo
    //commentaire sur une ligne
    /* Commentaire sur
       plusieurs lignes */
Constantes
    const1<-20 : entier
    const2<-"bonjour" : chaîne
Variables
    var1, var2 : entier
    var3 : chaîne
Début //corps de l'algorithme

    /*
    INSTRUCTIONS
    */

Fin
```

AFFECTATION : EXERCICES

Donnez les valeurs des variables A, B et C après exécution des instructions suivantes ?

Variables

A, B, C : entier

Début

A <- 7

B <- 17

A <- B

B <- A+5

C <- A+B

C <- B-A

Fin

AFFECTATION : EXERCICES

Donnez les valeurs des variables A et B après exécution des instructions suivantes ?

Variables

A, B : entier

Début

A <- 6

B <- 2

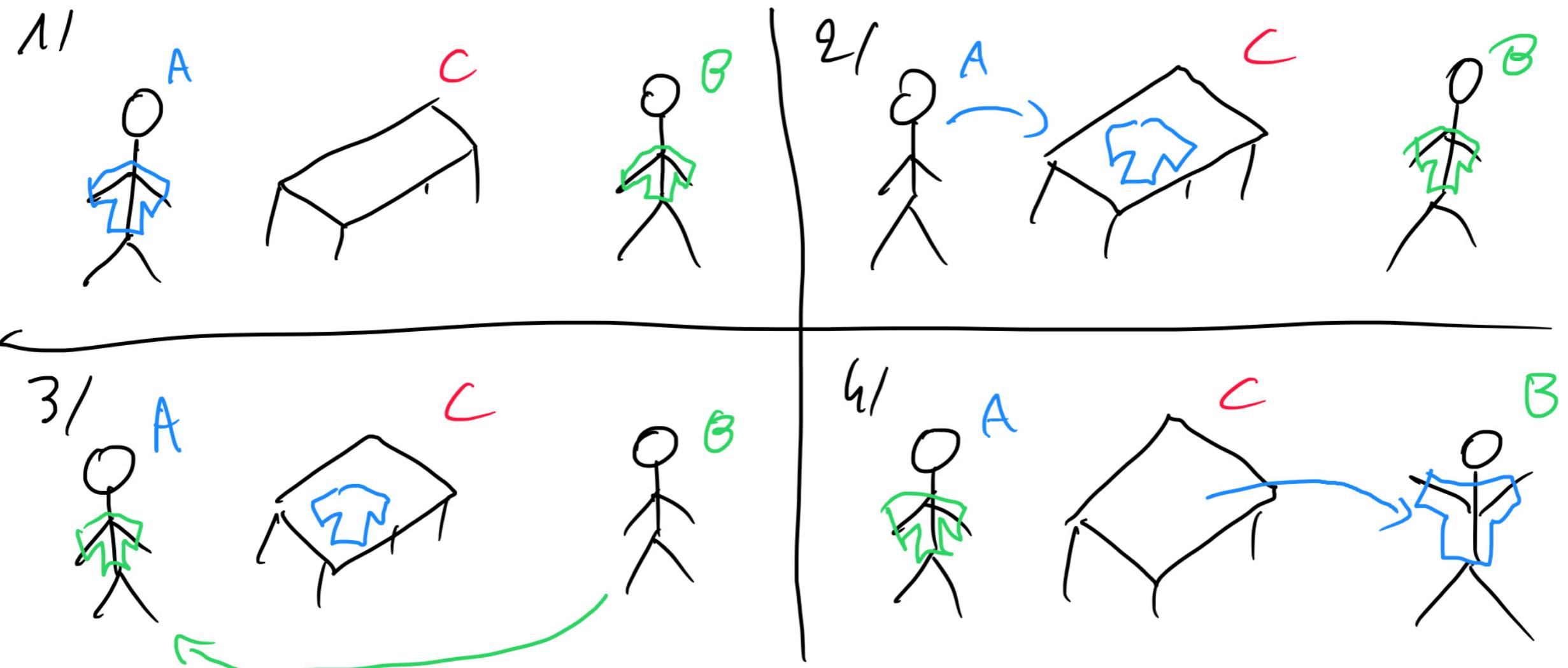
A <- B

B <- A

Fin

Les deux dernières instructions permettent-elles d'échanger les valeurs de A et B ?

AFFECTATION : ÉCHANGES



AFFECTATION : ÉCHANGES

Ecrire un algorithme permettant d'échanger les valeurs de deux variables A et B ?

EXPRESSIONS ET OPÉRATEURS

- Une expression peut être une valeur, une variable ou une opération constituée de variables reliées par des opérateurs
- Exemples : 1, b, a², a+3*b-c, ...
- L'évaluation de l'expression fournit une valeur unique qui est le résultat de l'opération
- Une expression est évaluée de gauche à droite mais en tenant compte des priorités des opérateurs.

EXPRESSIONS ET OPÉRATEURS

- Les opérateurs dépendent du type de l'opération, ils peuvent être :
 - ➔ des opérateurs arithmétiques: +, -, *, /, % (modulo)
 - ➔ des opérateurs logiques: NON (!), OU (||), ET (&&)
 - ➔ des opérateurs relationnels: =, <, >, <=, >=
 - ➔ des opérateurs sur les chaînes: & (concaténation)

PRIORITÉ DES OPÉRATEURS

- Pour les opérateurs arithmétiques, l'ordre de priorité est le suivant :
 - () : les parenthèses
 - \wedge : (élévation à la puissance)
 - * , / (multiplication, division)
 - + , - (addition, soustraction)
- Exemple : $9 + 3 * 4 = ?$
- Exemple : $(9 + 3) * 4 = ?$

LES OPÉRATEURS BOOLÉENS

LA TABLE DE VÉRITÉ...

a	b	a ET b	a OU b	NON a
vrai	vrai	vrai	vrai	fause
vrai	fause	fause	vrai	fause
fause	vrai	fause	vrai	vrai
fause	fause	fause	fause	vrai

contraire
de

a **ET** b
x

vrai \rightarrow 1
fause \rightarrow 0

a **OU** b
+

LES INSTRUCTIONS DE LECTURE

- La lecture permet d'entrer des données à partir du clavier
- En algo, on écrit : Lire (maVariable)
- La machine met la valeur entrée au clavier dans la variable nommée maVariable

LES INSTRUCTIONS D'ÉCRITURE

- L'écriture permet d'afficher des résultats à l'écran (ou de les écrire dans un fichier)
- En algo, on écrit : `Ecrire ("Bonjour")`
- Exemple : `Ecrire ("Bonjour", prenom, nom)`
- La machine affiche les valeurs des expressions de la liste.

LECTURE ET ÉCRITURE : EXEMPLE

- Ecrire un algorithme qui demande un nombre entier à l'utilisateur, puis qui calcule et affiche le carré de ce nombre

Algorithme CalculCarre

Variables

A, B : entier

Début

Ecrire("entrer la valeur de A")

Lire(A)

B \leftarrow A*A

Ecrire("le carre de ", A, "est :", B)

Fin

LECTURE ET ÉCRITURE : EXERCICE

- Écrire un algorithme qui permet d'effectuer la saisie d'un nom, d'un prénom et affiche ensuite le nom complet

LECTURE ET ÉCRITURE : EXERCICE 2

- Ecrire un programme qui lit le prix HT d'un article, le nombre d'articles et le taux de TVA.
- Le programme fournit dans un deuxième temps le prix total TTC correspondant.

INSTRUCTIONS CONDITIONNELLES

- Une condition est une expression écrite entre parenthèse à valeur booléenne.
- Les instructions conditionnelles servent à exécuter une instruction ou une séquence d'instructions que si une condition est vérifiée.

INSTRUCTIONS CONDITIONNELLES

- En algo, une condition s'écrit :

```
Si (condition) alors
    instruction ou suite d'instructions1
Sinon
    instruction ou suite d'instructions2
FinSi
```

INSTRUCTIONS CONDITIONNELLES

EXERCICE

Écrire un algorithme qui demande un nombre entier à l'utilisateur, puis qui teste et affiche s'il est supérieur à 12 ou non

INSTRUCTIONS CONDITIONNELLES IMBRIQUÉES

Les instructions conditionnelles peuvent avoir un degré quelconque d'imbrications

```
Si (condition1) alors
    Si (condition2) alors
        instructionsA
    Sinon
        instructionsB
    Finsi
Sinon
    Si (condition3) alors
        instructionsC
    Finsi
Finsi
```

INSTRUCTIONS CONDITIONNELLES IMBRIQUÉES: EXEMPLE

Variable n : entier

Début

Ecrire ("entrez un nombre : ")

Lire (n)

Si (n < 0) **alors**

Ecrire ("Ce nombre est négatif")

Sinon

Si (n = 0) **alors**

Ecrire ("Ce nombre est nul")

Sinon

Ecrire ("Ce nombre est positif")

Finsi

Finsi

Fin

INSTRUCTIONS CONDITIONNELLES IMBRIQUÉES : EXERCICE

Le prix de la clé USB dans un magasin spécialisé varie selon le volume acheté :

5€ l'unité si le nombre de clé USB à acheter est inférieur à 10,

4€ l'unité si le nombre de clé USB à acheter est compris entre 10 et 20,

3€ l'unité si le nombre de clé USB à acheter est au-delà de 20.

Écrivez un algorithme qui demande à l'utilisateur le nombre de clé USB qu'il souhaite acheter et qui calcule et affiche le prix à payer.

INSTRUCTIONS CONDITIONNELLES COMPOSÉES

- Une condition composée est une condition formée de plusieurs conditions simples reliées par des opérateurs logiques : ET, OU et NON
- Exemple : x compris entre 2 et 6 : $(x \geq 2)$ ET $(x \leq 6)$
- L'évaluation d'une condition composée se fait selon des règles présentées précédemment dans la table de vérité

INSTRUCTIONS CONDITIONNELLES COMPOSÉES : EXERCICE

Ecrire un algorithme qui demande trois noms à l'utilisateur et l'informe ensuite s'ils sont rangés ou non dans l'ordre alphabétique.

INSTRUCTION CAS

- Lorsque l'on doit comparer une même variable avec plusieurs valeurs, on peut remplacer cette suite de si par l'instruction cas

Cas où maVariable vaut

valeur1 : action1

valeur2 : action2

...

valeurn : actionn

autre : action autre

FinCas

INSTRUCTION CAS : EXEMPLE

Variables saison : chaîne

Début

Ecrire("Entrez une saison")

Lire (saison)

Cas où saison vaut

"Hiver": Ecrire("il fait froid")

"Ete": Ecrire("il fait chaud")

autre : Ecrire("je ne sais pas")

Fincas

Fin

INSTRUCTION CAS : EXEMPLE 2

Variables c : caractère

Début

Ecrire("entrer un caractère")

Lire (c)

Si((c>='A') et (c<='Z')) alors

Cas où c vaut

'A', 'E', 'I', 'O', 'U', 'Y' :

Ecrire("c'est une voyelle majuscule")

autre : Ecrire("c'est une consonne majuscule")

Fincas

Sinon

Ecrire("ce n'est pas une lettre majuscule")

Finsi

Fin

INSTRUCTIONS ITÉRATIVES :

LES BOUCLES

- Les boucles servent à répéter l'exécution d'un groupe d'instructions un certain nombre de fois
- On distingue trois sortes de boucles en langages de programmation :
 - ➔ Les boucles tant que : on y répète des instructions tant qu'une certaine condition est réalisée
 - ➔ Les boucles jusqu'à : on y répète des instructions jusqu'à ce qu'une certaine condition soit réalisée
 - ➔ Les boucles pour ou avec compteur : on y répète des instructions en faisant évoluer un compteur (variable particulière) entre une valeur initiale et une valeur finale

LA BOUCLE TANT QUE

TantQue (condition)

instructions

FinTantQue

- La condition (dite condition de contrôle de la boucle) est évaluée avant chaque itération
 - ➔ Si la condition est vraie, on exécute les instructions (corps de la boucle), puis, on retourne tester la condition. Si elle est encore vraie, on répète l'exécution.
 - ➔ Si la condition est fausse, on sort de la boucle et on exécute l'instruction qui est après FinTantQue

LA BOUCLE TANT QUE

- Une des instructions du corps de la boucle doit absolument changer la valeur de la condition de vrai à faux (après un certain nombre d'itérations), sinon le programme va tourner indéfiniment
- C'est ce qu'on appelle une boucle infinie



```
i<-1  
TantQue (i>0)  
    i<-i+1  
FinTantQue
```



LA BOUCLE TANT QUE : EXEMPLE

Contrôle de saisie d'une lettre alphabétique jusqu'à ce que le caractère entré soit valable

Variable C : caractère

Debut

 Ecrire ("Entrez une lettre majuscule")

 Lire (C)

TantQue (C < 'A' ou C > 'Z')

 Ecrire ("Saisie erronée. Recommencez")

 Lire (C)

FinTantQue

 Ecrire ("Saisie valable")

Fin

LA BOUCLE RÉPÉTER ... JUSQU'À

Répéter
instructions
Jusqu' à (condition)

- La condition est évaluée après chaque itération
- Les instructions entre Répéter et Jusqu'à sont exécutées au moins une fois et leur exécution est répétée jusqu'à ce que la condition soit vraie (tant qu'elle est fausse)

LA BOUCLE TANT QUE : EXERCICE

Ecrire un algorithme qui demande à l'utilisateur un nombre compris entre 1 et 3 jusqu'à ce que la réponse convienne.

LA BOUCLE TANT QUE : EXERCICE

Variable N : entier

Début

Ecrire("Entrez un nombre entre 1 et 3")

Lire(N)

TantQue (N < 1 OU N > 3)

Ecrire("Saisie erronée. Recommencez")

Lire(N)

FinTantQue

Ecrire("Good job")

Fin

LA BOUCLE POUR

Pour compteur **allant de** début **à fin faire**
instructions
FinPour

- Compteur est une variable de type entier. (elle doit être déclarée)
- Début et fin peuvent être des valeurs, des variables définies avant le début de la boucle ou des expressions de même type que compteur

LA BOUCLE POUR : EXEMPLE

```
Pour i allant de 0 à 10 faire  
    Ecrire(i)  
FinPour
```

Cette boucle affichera successivement les nombres 0, 1, ..., 10, on effectue donc 11 itérations.

LA BOUCLE POUR : EXERCICE

Ecrire l'algorithme permettant d'afficher la table de multiplication par 9.

LA BOUCLE POUR : EXERCICE 2

Ecrire un algorithme qui demande successivement 20 nombres à l'utilisateur, et qui lui dit ensuite quel était le plus grand parmi ces 20 nombres.

LA BOUCLE POUR : EXERCICE 3

Ecrire 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 4, le programme doit calculer:

$$1 + 2 + 3 + 4 = 10$$

BOUCLES IMBRIQUÉES

Les instructions d'une boucle peuvent être des instructions itératives.
Dans ce cas, on aboutit à des boucles imbriquées.

```
Pour i allant de 1 à 5
    Pour j allant de 1 à i
        Ecrire("O")
    FinPour
    Ecrire("K")
    Ecrire("\n")
FinPour
```

CHOIX D'UN TYPE DE BOUCLE

- Si on peut déterminer le nombre d'itérations avant l'exécution de la boucle, il est plus naturel d'utiliser la boucle Pour
- S'il n'est pas possible de connaître le nombre d'itérations avant l'exécution de la boucle, on fera appel à l'une des boucles TantQue ou répéter jusqu'à

LES FONCTIONS

SYNTAXE D'UNE FONCTION

Fonction nomDeLaFonction (Liste des paramètres) : Type de résultat

Début

Instructions de la fonction

Retourner (résultat)

Fin

Les paramètres sont facultatifs, mais s'il n'y pas de paramètres, les parenthèses doivent rester présentes.

L'instruction Retourner renvoie au programme appelant le résultat de l'expression placée à la suite de ce mot clé.

FONCTION : EXEMPLE

Calcul du périmètre d'un rectangle

Fonction perimetreRectangle (largeur,

longueur : entier) : entier

Variable perimetre : entier

Début

perimetre<-(2* (largeur+longueur))

Retourner (perimetre)

Fin

APPEL D'UNE FONCTION

nomDeLaFonction(Liste des paramètres)

Pour exécuter une fonction, il suffit de faire appel à elle en écrivant son nom suivie des paramètres effectifs.

Variable p : entier

Début

p <- **perimetreRectangle**(10, 15)

Ecrire("Le périmètre du rectangle
est : ", p)

Fin

PORTEE DES VARIABLES

- Une variable déclarée dans la partie déclaration de l'algorithme principale est appelée variable globale. Elle est accessible de n'importe où dans l'algorithme, même depuis les fonctions. Elle existe pendant toute la durée de vie du programme.
- Une variable déclarée à l'intérieur d'une fonction est dite locale. Elle n'est accessible que dans cette fonction, les autres fonctions n'y ont pas accès. La durée de vie d'une variable locale est limitée à la durée d'exécution de la fonction.

FONCTION : EXERCICE

Définir une fonction qui renvoie le plus grand de deux nombres différents.

Ecrire un programme qui demande deux nombres à l'utilisateur et qui affiche le plus grand des deux nombres en appelant la fonction précédemment créée.

FONCTION : EXERCICE

```
//Déclaration de la fonction Max  
Fonction Max(X: entier, Y: entier) : entier  
Début  
    Si X > Y Alors  
        Retourner X  
    Sinon  
        Retourner Y  
    FinSi  
Fin
```

```
//Algorithme principal  
Variables A, B, M : entier
```

```
Début  
    Ecrire("Donnez la valeur de A :")  
    Lire(A)  
    Ecrire("Donnez la valeur de B :")  
    Lire(B)  
    //Appel de la fonction Max  
    M ← Max(A,B)  
    Ecrire(« Le plus grand de ces deux nombres est : », M)  
Fin
```

LES TABLEAUX

PROBLÉMATIQUE

Algorithme Note

Variables N1, N2, N3, N4, N5 : Réel

Début

```
Ecrire("Entrer la valeur de la 1er note")
Lire(N1)
Ecrire("Entrer la valeur de la 2eme note")
Lire(N2)
Ecrire("Entrer la valeur de la 3eme note")
Lire(N3)
Ecrire("Entrer la valeur de la 4eme note")
Lire(N4)
Ecrire("Entrer la valeur de la 5eme note")
Lire(N5)
Ecrire("La note 1 multipliée par 3 est : ",N1 * 3)
Ecrire("La note 2 multipliée par 3 est : ",N2 * 3)
Ecrire("La note 3 multipliée par 3 est : ",N3 * 3)
Ecrire("La note 4 multipliée par 3 est : ",N4 * 3)
Ecrire("La note 5 multipliée par 3 est : ",N5 * 3)
```

Fin

DÉFINITION D'UN TABLEAU

- Un tableau est une suite d'éléments de même type.
- Il utilise plusieurs cases mémoire à l'aide d'un seul nom.
- Comme toutes les cases portent le même nom, elles se différencient par un numéro ou un indice.

SYNTAXE D'UN TABLEAU

Variable identifiant :
tableau [taille] **de type**

- La taille du tableau doit être un nombre entier.
- Les éléments d'un tableau sont des variables qui s'utilisent exactement comme n'importe quelles autres variables classiques.

UTILISATION D'UN TABLEAU

- Soit le tableau suivant : Variable Note : tableau [30] de réels
- L'utilisation de ces éléments se fait via le nom du tableau et son indice. Cet indice peut être soit une valeur (exemple : Note [3]), soit une variable (exemple : Note [i]) ou encore une expression (exemple : Note [i+1]).
- Le premier élément d'un tableau porte l'indice zéro.

UTILISATION D'UN TABLEAU :

EXEMPLES

- Exemple 1 : L'instruction suivante affecte à la variable X la valeur du premier élément du tableau Note :
 - $X \leftarrow \text{Note}[0]$
- Exemple 2 : L'instruction suivante affiche le contenu de l'élément en quatrième position du tableau Note :
 - Ecrire ($\text{Note}[4]$)
- Exemple 3 : L'instruction suivante affecte une valeur introduite par l'utilisateur à la position trois du tableau Note :
 - Lire ($\text{Note}[3]$)

UTILISATION D'UN TABLEAU :

EXERCICE

Ecrire un algorithme qui déclare et stocke dans un tableau les six voyelles de l'alphabet

UTILISATION D'UN TABLEAU :

EXERCICE

```
Algorithme tableau_voyelles
    Variable voyelles : tableau[6] de Caractère
    Début
        voyelles[0] <- 'a'
        voyelles[1] <- 'e'
        voyelles[2] <- 'i'
        voyelles[3] <- 'o'
        voyelles[4] <- 'u'
        voyelles[5] <- 'y'
    Fin
```

UTILISATION D'UN TABLEAU :

EXERCICE 2

Écrire un algorithme permettant de saisir 30 notes et de les afficher.

UTILISATION D'UN TABLEAU :

EXERCICE 2

Algorithme tableau_note

Variable notes : tableau[30] de réels, i : entier

Début

//Remplissage du tableau notes

Pour i Allant de 0 à 29 Faire

Ecrire("Entrer la valeur de la note n°", i+1)

Lire(notes[i])

FinPour

//Affichage des notes

Pour i Allant de 0 à 29 Faire

Ecrire("La note ", i+1, " vaut ", notes[i])

FinPour

Fin

TABLEAUX À 2 DIMENSIONS

- Les tableaux à deux dimensions se représentent comme une matrice ayant un certain nombre de lignes (première dimension) et un certain nombre de colonnes (seconde dimension).
- Reprenons l'exemple des notes en considérant cette fois qu'un étudiant a plusieurs notes (une note pour chaque matière). Pour quatre étudiants, nous aurions le tableau de relevés des notes suivant :

	Etudiant 1	Etudiant 2	Etudiant 3	Etudiant 4
Informatique	12	13	9	10
Comptabilité	12.5	14	12	11
Mathématiques	15	12	10	13

SYNTAXE D'UN TABLEAU À 2 DIMENSIONS

Variable identificateur :
tableau [nb_lignes, nb_colonnes] **de type**

L'instruction suivante déclare un tableau Note de type réel à deux dimensions composé de 3 lignes et de 4 colonnes :

Variable Note : **tableau** [3, 4] **de réels**

UTILISATION D'UN TABLEAU À 2 DIMENSIONS

- Pour accéder à un élément de la matrice (tableau à deux dimensions), il suffit de préciser, entre crochets, les indices de la case contenant cet élément.
- Exemple : L'instruction suivante affecte à la variable X la valeur de l'élément du tableau Note positionné sur la première ligne et la première colonne :

→ X ← Note[0, 0]

PARCOURS D'UN TABLEAU

À 2 DIMENSIONS

- Pour parcourir une matrice nous avons besoin de deux boucles, l'une au sein de l'autre, c'est ce qu'on appelle les boucles imbriquées.
- La première boucle est conçue pour parcourir les lignes.
- Tandis que la deuxième est utilisée pour parcourir les colonnes de la ligne précisée par la boucle principale (la première boucle).

TABLEAU À 2 DIMENSIONS : EXERCICE

Écrire un algorithme permettant la saisie des notes d'une classe de 15 étudiants pour 3 matières.

TABLEAU À 2 DIMENSIONS :

EXERCICE

Algorithme Notes

Constantes N = 15, M = 3 : entier

Variables note : tableau[N, M] de réels

i, j : entier

Début

//Remplissage du tableau note

Pour i Allant de 0 à N-1 Faire

Pour j Allant de 0 à M-1 Faire

Ecrire("Entrer la note de l'étudiant", i+1,

" dans la matière ", j+1)

Lire(note[i, j])

FinPour

FinPour

Fin

TABLEAU DYNAMIQUE

- Les tableaux définis jusqu'ici sont dits statiques, car il faut qu'au moment de l'écriture du programme le développeur décide de la taille maximale que pourra atteindre le tableau.
- Dans certains situations, on ne peut pas savoir la taille du tableau dans la phase de programmation.
- Les tableaux dynamiques sont des tableaux dont la taille n'est définie que lors de l'exécution. Pour créer un tableau dynamique, il suffit de lui affecter une taille vide.

SYNTAXE D'UN TABLEAU DYNAMIQUE

Variable identifiant : **tableau [] de type**

Comme un tableau dynamique ne possède pas de taille prédéfinie, il convient de redimensionner le tableau avant de pouvoir s'en servir.

Redimensionner(identifiant[taille])

TABLEAU DYNAMIQUE : EXEMPLE

Variables

nb en entier

Note[] de type entiers

Début

Ecrire("Combien y a-t-il de notes à saisir ?")

Lire(nb)

Redimensionner(Note[nb])

...

Fin

TABLEAU DYNAMIQUE :

EXERCICE

Ecrire un algorithme constituant un tableau, à partir de deux tableaux de même longueur préalablement saisis.

Le nouveau tableau sera la somme des éléments des deux tableaux de départ.

6	4	5	1	9
---	---	---	---	---

TABLEAU 1

4	7	3	1	5
---	---	---	---	---

TABLEAU 2

10	11	8	2	14
----	----	---	---	----

TABLEAU 3

TABLEAU DYNAMIQUE : EXERCICE

Algorithme TableauAddition

Variables

nb, i : entier;

T1, T2, T3 : tableau[] de type entier;

Début

Ecrire("Longueur des tableaux ?")

Lire(nb)

Redimensionner T1 [nb]

Redimensionner T2 [nb]

Redimensionner T3 [nb]

/* Remplissage tableau 1 et 2 */

Pour i de 0 à nb - 1 faire

 T3[i] <- T1[i] + T2[i];

FinPour

Fin

PROBLÉMATIQUE

On veut représenter dans un programme les données de 20 personnes avec pour chacune un nom, un prénom et un âge.

```
nom1, prenom1 : chaine  
age1 : entier  
...  
nom20, prenom20 : chaine  
age20 : entier  
...  
nom1 <- "Dupont";  
prenom1 <- "Jean";  
age1 <- 34;  
...
```

TABLEAU ASSOCIATIF OU ENREGISTREMENT

- Vous connaissez les tableaux. Ils permettent de mémoriser des informations et d'y accéder par leur index.
- Un tableau associatif fait sensiblement la même chose, mais au lieu d'utiliser des index numériques pour retrouver les valeurs, il permet d'utiliser n'importe quel autre type de donnée.
- Dans ce contexte, on parle de clé plutôt que d'index.
- En Algorithmique on parle d'un enregistrement

DÉCLARATION D'UN ENREGISTREMENT

Enregistrement identificateur

clé1: type,

clé2: type,

...

FinEnregistrement

DÉCLARATION D'UN ENREGISTREMENT EXEMPLE

Enregistrement Personne

nom: chaine,

prenom: chaine,

age: entier

FinEnregistrement

INITIALISATION D'UN ENREGISTREMENT

- Il faut créer une "instance" de l'enregistrement Personne pour pouvoir l'utiliser.
- On peut créer autant d'instance que nécessaire.
- Fonctionne comme une variable classique, l'identificateur de l'enregistrement devient un nouveau type de variable

Variable p1: Personne

REmplissage d'un enregistrement

Deux possibilités pour remplir une instance d'un enregistrement

```
p1.nom <- "Dupont"  
p1.prenom <- "Jean"  
p1.age <- 34
```

ou

```
p1 <- {  
  nom: "Dupont",  
  prenom: "Jean",  
  age: 34  
}
```

ENREGISTREMENTS : EXERCICE

Écrire un enregistrement permettant de stocker les informations sur un véhicule chez un concessionnaire

ENREGISTREMENTS :

EXERCICE

Écrire un enregistrement permettant de stocker les informations sur un véhicule chez un concessionnaire

Enregistrement Voiture

modele: chaine,

marque: chaine,

annee: entier,

kilometrage: entier

FinEnregistrement

ENREGISTREMENTS : EXERCICE

Stocker le véhicule de votre choix dans une instance de l'enregistrement précédemment créé

ENREGISTREMENTS : EXERCICE

Stocker le véhicule de votre choix dans une instance de l'enregistrement précédemment créé

Variable v1: Voiture

```
v1.modele <- "Mégane"  
v1.marque <- "Renault"  
v1.annee <- 2017  
v1.kilometrage <- 55000
```

ENREGISTREMENTS : EXERCICE

Stocker le véhicule de votre choix dans une instance de l'enregistrement précédemment créé

```
Variable v1: Voiture  
v1 <- {  
    modele: "Mégane",  
    marque: "Renault",  
    annee: 2017,  
    kilometrage: 55000  
}
```

ENCORE UNE PROBLÉMATIQUE...

On veut toujours stocker dans un programme les données de 20 personnes avec pour chacune un nom, un prénom et un âge.

Enregistrement Personne

nom: chaine,

prenom: chaine,

age: entier

FinEnregistrement

Variables

p1, p2, p3, ..., p20: **Personne**

TABLEAU D'ENREGISTREMENTS

- Vous connaissez les tableaux. Ils permettent de mémoriser des informations et d'y accéder par leur index.
- Nous avons vu jusqu'à maintenant que dans un tableau on pouvait mettre des entiers, des chaînes, etc...
- Nous avons vu via la déclaration d'un enregistrement que cela permettait d'avoir un « nouveau » type à disposition
- Donc nous pouvons créer des tableaux stockant des enregistrements d'un certain type prédéfinie.

DÉCLARATION D'UN TABLEAU D'ENREGISTREMENTS

variable

personnes : tableau [20] de Personne

INITIALISATION D'UN TABLEAU D'ENREGISTREMENTS

```
personnes[0].nom <- "Dupont"  
personnes[0].prenom <- "Jean"  
personnes[0].age <- 34  
  
...  
personnes[19].nom <- "Dupond"  
personnes[19].prenom <- "Julien"  
personnes[19].age <- 26
```

TABLEAU D'ENREGISTREMENTS

EXERCICE

- 1/ Créer un tableau d'enregistrements pouvant contenir 10 instances de Voiture
- 2/ Utiliser une boucle pour parcourir ce tableau et le remplir par les données saisies par l'utilisateur

MÉMO

```
Enregistrement Voiture
    modele: chaine,
    marque: chaine,
    annee: entier,
    kilometrage: entier
FinEnregistrement
```

TABLEAU D'ENREGISTREMENTS

EXERCICE

Variables voitures: tableau[10] de Voiture, i: entier

Début

Pour i allant de 0 à 9 faire

Ecrire("Saisir la marque de la voiture n°", i+1)
Lire(voitures[i].marque)

Ecrire("Saisir le modèle de la voiture n°", i+1)
Lire(voitures[i].modele)

Ecrire("Saisir l'année de la voiture n°", i+1)
Lire(voitures[i].annee)

Ecrire("Saisir le kilométrage de la voiture n°", i+1)
Lire(voitures[i].kilometrage)

FinPour

Fin

EXERCICES COMPLÉMENTAIRES

EXERCICES COMPLÉMENTAIRES

PRIORITÉ DES OPÉRATEURS

Donner les valeurs des variables a, b, c et d

```
a <- 4 * 2 + 5
b <- 5 + 3 * 2 - 6
c <- (a > b) ET (b > 2)
d <- (a < b) OU (b > 2)
```

EXERCICES COMPLÉMENTAIRES

PRIORITÉ DES OPÉRATEURS

Donner les valeurs des variables a, b, c et d

```
a <- 4 * 2 + 5  
b <- 5 + 3 * 2 - 6  
c <- (a > b) ET (b > 2)  
d <- (a < b) OU (b > 2)
```

SOLUTION :

A = 13, B = 5, C = VRAI ET D = VRAI.

EXERCICES COMPLÉMENTAIRES

TRIANGLE D'ÉTOILES

Ecrire un algorithme qui affiche à l'écran le triangle d'étoiles suivant :

```
*  
**  
***  
****  
*****
```

EXERCICES COMPLÉMENTAIRES

TRIANGLE D'ÉTOILES

Algorithm TriangleEtoiles

Variables i, j : entiers

Début

Pour i Allant de 1 à 5 Faire

Pour j Allant de 1 à i faire
 Ecrire ("*")

FinPour

 //Retour à la ligne
 Ecrire ("\n")

FinPour

Fin

EXERCICES COMPLÉMENTAIRES

DEVINETTE

Ecrire un programme mettant en oeuvre le jeu suivant :

Le premier utilisateur saisi un entier que le second doit deviner. Pour cela, il a le droit à autant de tentatives qu'il souhaite.

A chaque échec, le programme lui indique si l'entier cherché est plus grand ou plus petit que sa proposition.

Un score indiquant le nombre de coups joués est mis à jour et affiché lorsque l'entier est trouvé.

EXERCICES COMPLÉMENTAIRES

DEVINETTE

Algo Devinette

Variables nb, reponse, score : entier

Début

Ecrire("Joueur 1 saisi un nombre à faire deviner")

Lire(nb)

score <- 0

Répéter

Ecrire("Joueur 2 tente ta chance")

Lire(reponse)

Si (reponse > nb) alors

Ecrire("Le nombre recherché est plus petit")

Sinon

Si (reponse < nb) alors

Ecrire("Le nombre recherché est plus grand")

FinSi

FinSi

score <- score + 1

Jusqu'à (nb = reponse)

Ecrire("Bonne réponse, chapeau l'artiste")

Ecrire(score, " tentatives pour trouver la réponse")

Fin

EXERCICES COMPLÉMENTAIRES

SUPPRIMER UN ÉLÉMENT

Ecrire un algorithme qui permet à un utilisateur de supprimer une valeur donnée stockée dans un tableau d'entiers.

EXERCICES COMPLÉMENTAIRES

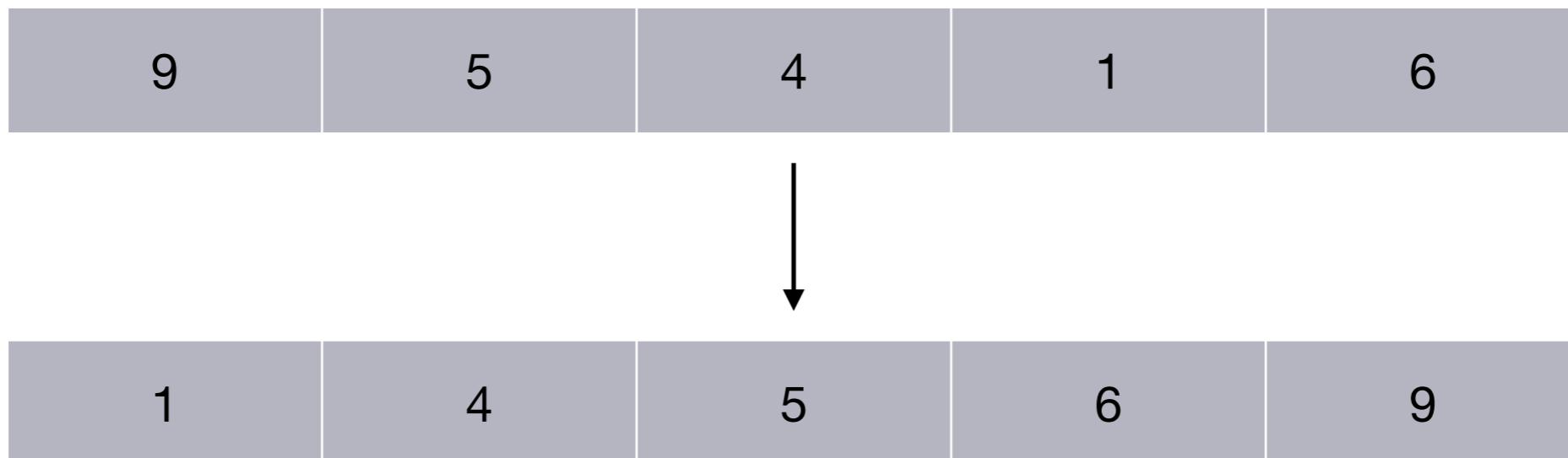
SUPPRIMER UN ÉLÉMENT

```
Algorithme SupprimerElement
Variables tab : tableau[ ] d'entiers
            i : entier
Début
    // Remplissage du tableau
    Ecrire("Entrez la valeur à supprimer :");
    Lire(elt);
Pour i Allant de 1 à Compteur(tab)-1 Faire
    Si elt = tab[i] alors
        tab[i] <- tab[Compteur(tab)-1]
    FinSi
FinPour
Redimensionner(tab[Compteur(tab)-1])
Fin
```

EXERCICES COMPLÉMENTAIRES

TRI D'UN TABLEAU

Écrire un algorithme permettant de trier un tableau



EXERCICES COMPLÉMENTAIRES

TRI D'UN TABLEAU

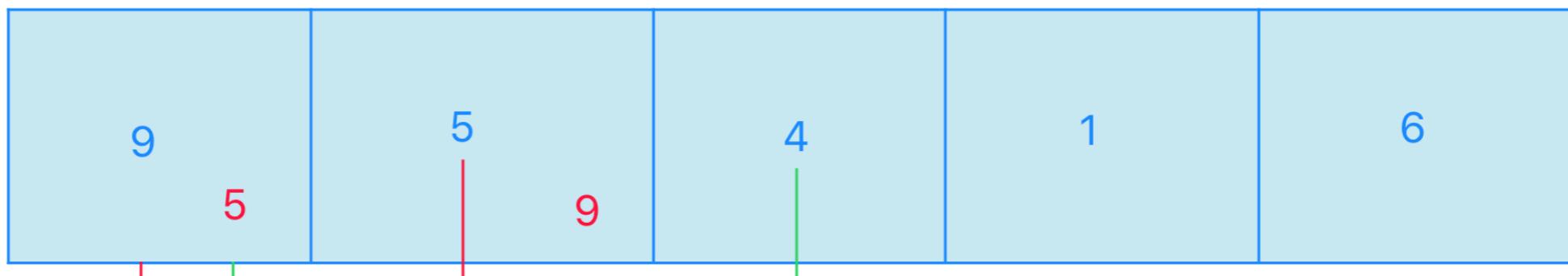
Écrire un algorithme permettant de trier un tableau

Indices :

- on cherche l'élément de la plus petite valeur dans le tableau
- on le place en tête du tableau
- recommencer avec le tableau moins la première case

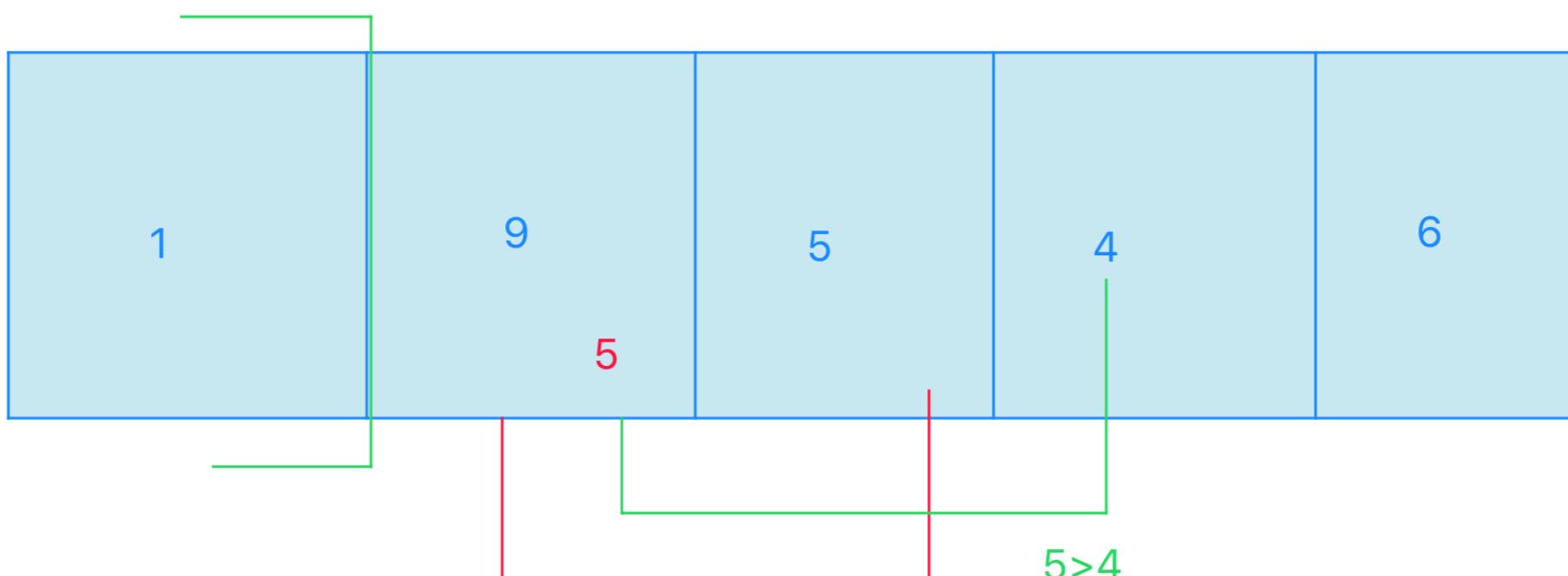
EXERCICES COMPLÉMENTAIRES

TRI D'UN TABLEAU



9>5

5>4



9>5

5>4

EXERCICES COMPLÉMENTAIRES

TRI D'UN TABLEAU

Algorithme TriTableau

Variables tab : tableau[5] d'entiers
i, j, tmp : entier

Début

// Remplissage du tableau

tab[0] <- 9 // etc...

// Tri du tableau

Pour i allant de 0 à Compteur(tab)-2 **Faire**

Pour j allant de i+1 à Compteur(tab)-1 **Faire**

Si (tab[j] < tab[i]) **alors**

tmp <- tab[i]

tab[i] <- tab[j]

tab[j] <- tmp

FinSi

FinPour

FinPour

Fin

EXERCICES COMPLETS

EXERCICES COMPLETS

DISTRIBUTEUR DE BOISSELS



EXERCICES COMPLETS

DISTRIBUTEUR DE BOISSONS

4 boissons : Coca Cola (Prix 0.80€), Fanta (Prix 0.70€), Ice Tea (Prix 0.70€), Perrier (Prix 1€)

1/ L'utilisateur doit saisir la boisson souhaitée

2/ Afficher le prix de la boisson souhaitée

3/ Tester si la somme payée est correcte

ATTENTION LE DISTRIBUTEUR REND LA MONNAIE

4/ Afficher un message « Votre boisson bidule est prête »

EXERCICES COMPLETS

PARKING SOUS TERRAIN

Tarifs :

15 Minutes 0,80 €,

30 Minutes 1,30 €,

45 Minutes 1,70 €,

Au delà de 45 Minutes tarif journée : 6€

On considère que la machine qui délivre les tickets et la machine pour payer partagent les informations sur les véhicules et le même algorithme

EXERCICES COMPLETS

PARKING SOUS TERRAIN

PARTIE COMMUNE

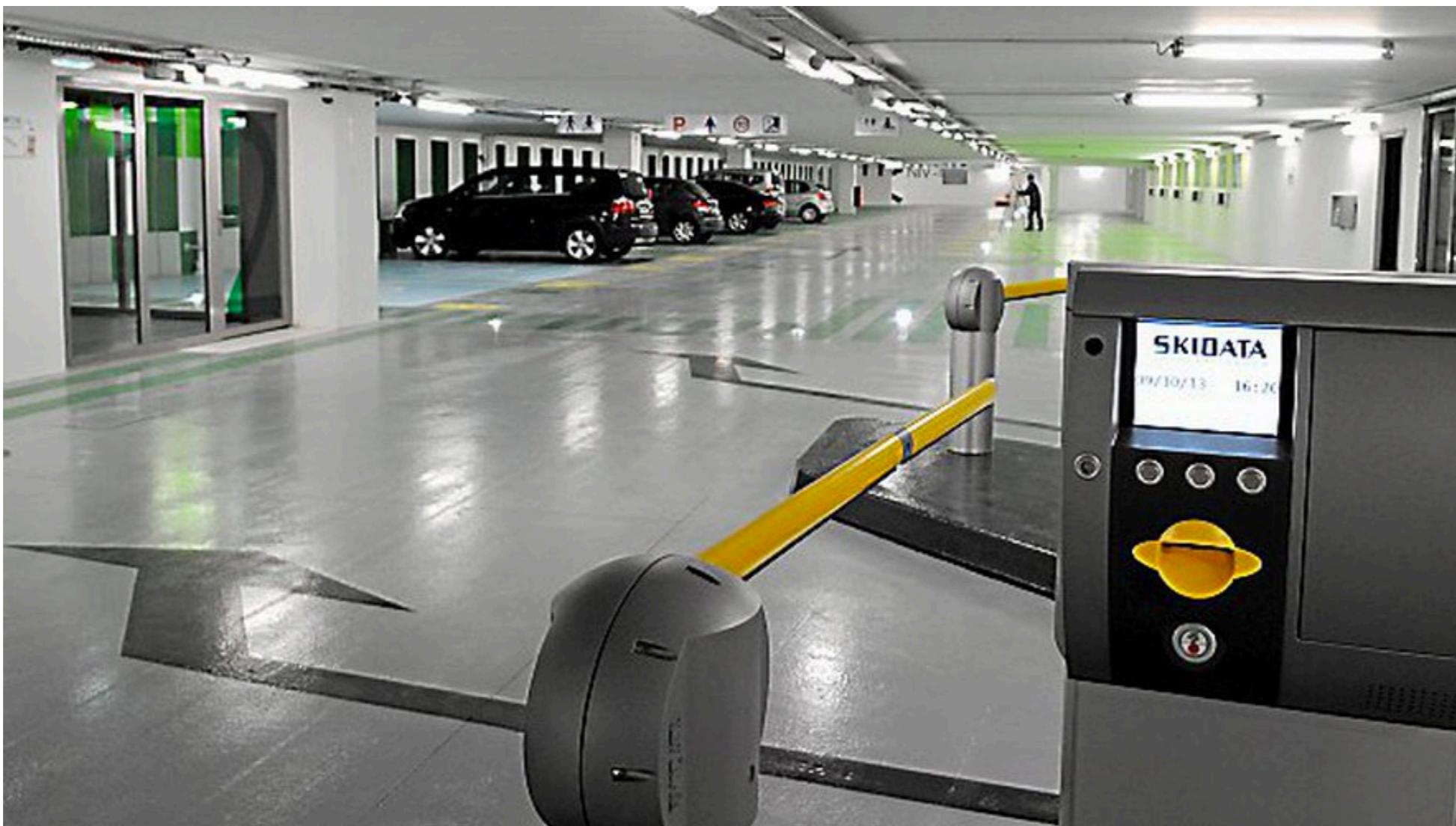
On considère que le programme partage un enregistrement Vehicule et un tableau d'enregistrements Vehicule.

Ces éléments sont initialisés une fois pour toute.

EXERCICES COMPLETS

PARKING SOUS TERRAIN

PARTIE 1 : DELIVRER LE TICKET



EXERCICES COMPLETS

PARKING SOUS TERRAIN

PARTIE 1 : DELIVRER LE TICKET

- 1/ L'utilisateur doit saisir sa plaque d'immatriculation
- 2/ On stocke l'heure d'arrivée de l'utilisateur et on lui délivre un ticket (on considère que l'on récupère l'heure courante avec une fonction que nous n'écrivons pas : getDate())
- 3/ Plusieurs utilisateurs peuvent utiliser le parking
- 4/ Prévoir une astuce pour attendre l'arrivée d'un véhicule sans que le programme ne s'arrête

EXERCICES COMPLETS

PARKING SOUS TERRAIN

PARTIE 2 : CAISSE



EXERCICES COMPLETS

PARKING SOUS TERRAIN

PARTIE 2 : CAISSE

- 1/ On ignore l'étape de décodage du ticket, pour plus de simplicité le numéro de la plaque sera saisi par l'utilisateur
- 2/ On stocke l'heure de retour
- 3/ On lui indique le prix à payer (on considère que nous avons une fonction `getDiffDate(date1, date2)` à notre disposition)

EXERCICES COMPLETS

GESTION D'UN HOTEL

Nous voulons gérer les entrées/sorties des clients à l'accueil de l'hôtel.



EXERCICES COMPLETS

GESTION D'UN HOTEL

Nous disposons de ces informations :

- Arrivée à partir de 15h
- Départ avant 11h
- Si arrivée prématuée on refuse le client
- Si départ après l'heure on majore la facture du client de 10€
- Choix du nombre de nuit à l'arrivée
- Paiement de la chambre au départ
- Option petit déjeuner pendant le séjour (7€/nuit)

EXERCICES COMPLETS

GESTION D'UN HOTEL

Pour simplifier une seule gamme de chambre est disponible au prix de 65€/nuit

On initialise notre programme avec deux clients déjà présent dans l'hôtel :

- José Garcia : 4 nuits / arrivée avant hier à 16h / pas de petit déjeuner
- Antoine de Caunes : 2 nuits / arrivée hier à 18h / option petit déjeuner

EXERCICES COMPLETS

GESTION D'UNE PIZZERIA

Les seules informations dont vous disposez :

- Commandes à emporter
- 4 pizzas à la carte de la pizzeria

A vous d'analyser le besoin et de trouver les solutions



VOUS POUVEZ DÉSORMAIS PASSER AU NIVEAU SUIVANT

APPRENDRE UN LANGAGE DE PROGRAMMATION 💪

