

Sommaire

Chapitre 1 : Introduction	1
1.1. Composants d'un ordinateur	1
Chapitre 2 : Algorithme.....	2
2.1. Introduction à la notion d'algorithme	2
2.1.1. Exemples simples d'exécution de programmes	2
2.1.2. Exemple d'algorithme.....	3
2.1.3. Définition d'un algorithme.....	3
2.1.4. Structure d'un algorithme	4
2.1.5. Les instructions de base ('Lire', 'Ecrire' et l'affectation).....	4
2.1.6. Algorithme vs Programme	5
2.1.7. Utilisateur vs Programmeur (Développeur)	5
2.1.8. Conventions syntaxiques de base.....	6
2.1.9. Déroulement manuel d'un algorithme	6
2.1.10. Questions de compréhension	7
2.1.11 Exercices	8
2.2. Algorithmes équivalents	9
2.2.1. Algorithme optimal (meilleur)	9
2.2.2. Messages indicatifs et explicatifs.....	9
2.2.3. Questions de compréhension	10
2.2.4. Exercices	10
2.3. Types de variables.....	12
2.3.1. Exercice	12
2.3. Traitement conditionnel (Alternatif)	13
2.3.1. Conventions syntaxiques des conditions.....	15
2.3.2. Equivalent du traitement conditionnel dans les langages de programmation	15
2.3.3. Traitements conditionnels séquentiels vs imbriqués	15
2.3.4. Traitement à choix multiple.....	17
2.3.5. Exercices	18
2.4. Traitement répétitif (Les boucles)	20
2.4.1. La boucle Pour	20
2.4.2. La boucle TantQue	21
2.4.3. La boucle Répéter	22

2.4.4. Boucles imbriqués.....	23
2.4.5. Passage entre les boucles	23
2.4.6. Choix du type de boucle	23
2.4.7. Quelques mécanismes algorithmiques répétitifs fondamentaux	24
2.4.8. Exercices	25

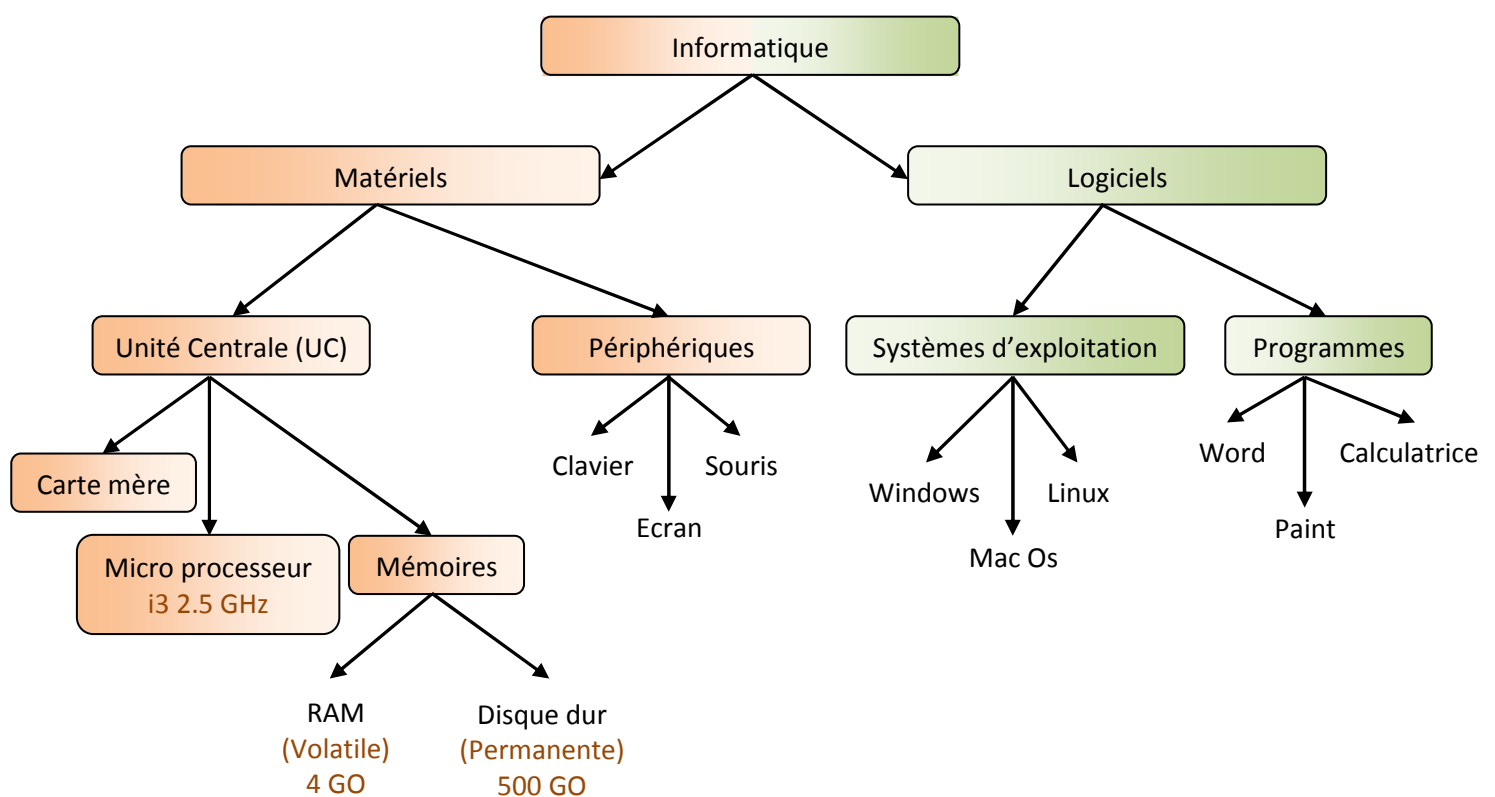
Chapitre 1 : Introduction

1.1. Composants d'un ordinateur

Le mot informatique est composé des deux termes : information et automatique. L'informatique donc est la science de traitement automatique de l'information.

On peut voir l'informatique selon deux aspects : matériel (hardware) et logiciel (software).

Le schéma suivant résume les composants matériels et logiciels d'un ordinateur :



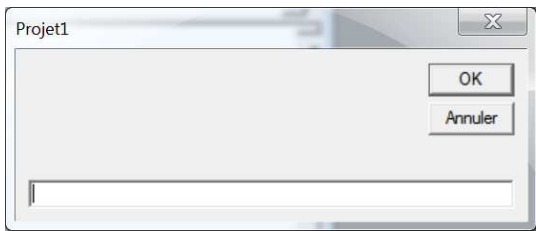
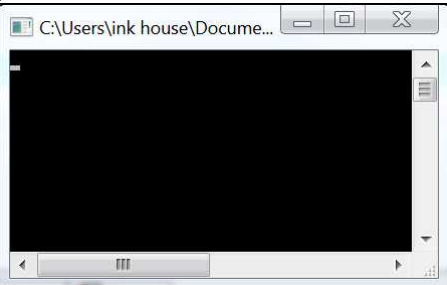
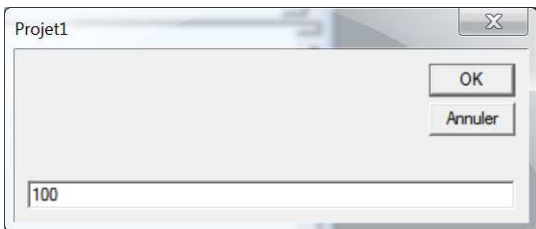
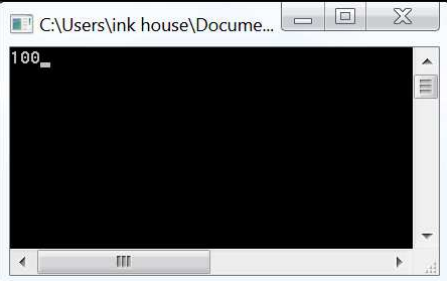
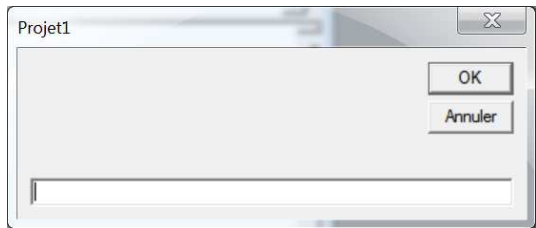
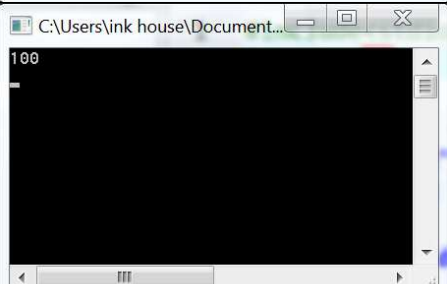
Chapitre 2 : Algorithme

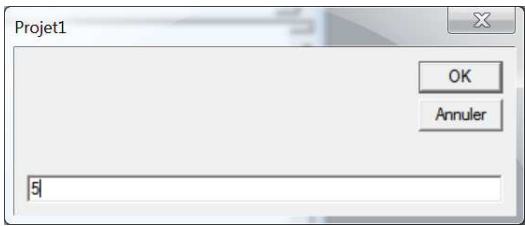
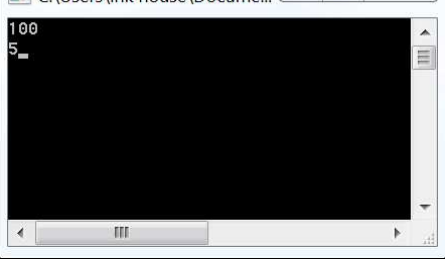
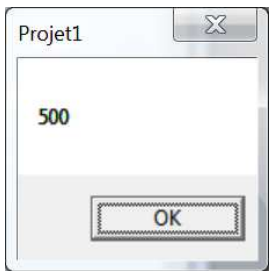
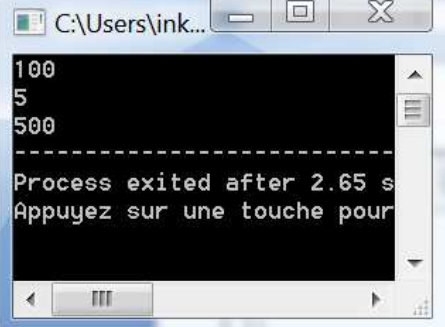
2.1. Introduction à la notion d'algorithme

Afin d'introduire la notion d'algorithme, nous allons présenter deux programmes très simples et décrire leurs exécutions sur machine.

2.1.1. Exemples simples d'exécution de programmes

Soit les deux programmes simples suivants, le premier en Visual Basic et le deuxième en langage C. Le tableau suivant décrit l'exécution du point de vue utilisateur, des deux programmes sur machine, étape par étape :

Exécution d'un programme en Visual Basic	Exécution d'un programme en C	Description des étapes d'exécution
<pre>Private Sub Form_Load() Dim a, b, c As Integer a = InputBox("") b = InputBox("") c = a * b MsgBox (c) End sub</pre>	<pre>int main() { int a, b, c; scanf("%d",&a); scanf("%d",&b); c = a*b; printf("%d", c); }</pre>	Code d'un programme
		Le programme affiche le curseur pour permettre à l'utilisateur d'entrer une valeur
		L'utilisateur saisit la valeur : 100
		Le programme affiche le curseur une autre fois

		l'utilisateur saisit une deuxième valeur : 5
		Le programme affiche le résultat de la multiplication des deux valeurs entrées par l'utilisateur : 500

2.1.2. Exemple d'algorithme

Un algorithme est l'équivalent d'un programme, il est écrit en pseudo code. L'algorithme suivant est l'équivalent en pseudo code des deux programmes de la section précédente :

Code en Visual Basic	Code en C	Pseudo Code	
Private Sub Form_Load()	main()	<u>Algo</u> Exo1	Entête
Dim a, b, c As Integer	int a, b, c;	<u>Variables</u> a,b,c : entier	Déclaration
		<u>Début</u>	Annoncer le début des instructions
a = InputBox ("")	scanf ("%d",&a);	Lire(a)	Permettre à l'utilisateur de saisir la valeur de a .
b = InputBox ("")	scanf ("%d",&b);	Lire(b)	Permettre à l'utilisateur de saisir la valeur de b .
c = a * b	c = a*b;	c ← a*b	Multiplier a par b et mettre le résultat dans c
MsgBox (c)	printf ("%d", c);	Ecrire(c)	Afficher le résultat stocké dans c
<u>End sub</u>	}	<u>Fin</u>	Annoncer la Fin

2.1.3. Définition d'un algorithme

Le mot algorithme vient du nom d'Al-Khwârizmî (الخوارزمي), grand mathématicien du 9^{ème} siècle.

Définition 1 : Un algorithme est une suite finie et non ambiguë d'opérations ou d'instructions permettant de résoudre un problème¹.

Définition 2 : Un algorithme est une méthode générale pour résoudre un type de problèmes. Il est dit correct lorsque, pour chaque instance du problème, il se termine en produisant la bonne sortie, c'est-à-dire qu'il résout le problème posé².

¹ Wikipedia

Définition 3 : Un algorithme, c'est tout simplement une façon de décrire dans ses moindres détails comment procéder pour faire quelque chose³.

2.1.4. Structure d'un algorithme

L'algorithme se compose de façon générale de trois blocs :

```
Algo Exo1 } Entête
Variables }
...        } Déclaration
Début    }
...        } Corps (Suite d'instructions)
Fin      }
```

1. Dans le bloc entête, on donne un nom à l'algorithme. Ce nom peut être composé de lettres et chiffres et doit obligatoirement commencer par une lettre, et ne doit pas contenir de lettres de ponctuation, l'espace en particulier.
2. Dans le bloc déclaration, on déclare les types de variables utilisées. Une variable est l'association d'un identifiant (un nom de variable) à une donnée. Une variable peut être du type **Entier** comme l'exemple introductif, **Réel** comme l'exemple de la section 2.1.5, **Caractère** et **chaîne de caractère** comme l'exemple de la section 2.3, et d'autres types qu'on verra au fur et à mesure de l'avancement dans ce cours.
3. Le bloc corps contient les instructions proprement dites, à exécuter dans l'ordre.

2.1.5. Les instructions de base ('Lire', 'Ecrire' et l'affectation)

Dans la quasi-totalité des algorithmes, on trouve les trois instructions de base : l'instruction **Lire**, l'instruction **Ecrire** et l'instruction d'**affectation**.

- L'instruction **Lire** permet de donner la main (permettre) à l'utilisateur d'entrer une ou plusieurs données.
- L'instruction **Ecrire** permet d'afficher (imprimer) des résultats ou des messages sur écran.
- L'instruction d'affectation (\leftarrow) permet de sauvegarder (ranger) une valeur dans une variable.

Exemple :

Soit l'algorithme suivant qui permet de calculer l'aire (la surface) d'un cercle à partir de son rayon.

```
Algo Aire
Variables
R, A : Réel
Début
Lire(R)
A  $\leftarrow$  3.14 * R * R
Ecrire(A)
Fin
```

Remarquez que le type des variables déclarées dans cet algorithme est **Réel**, puisque le rayon peut être en virgule comme 2.5, et le résultat est obligatoirement en virgule du fait de la multiplication par 3.14.

Cet algorithme permet de :

1. Donner la main à l'utilisateur pour **entrer** une valeur réelle du rayon d'un cercle.
2. Calculer l'aire (la superficie) à partir de la formule πR^2 , et ranger le résultat dans la variable **A** à travers l'instruction d'affectation (\leftarrow) qu'on lit : **reçoit** (A reçoit $3.14 * R * R$).
3. Afficher le résultat (l'aire calculé du cercle).

² Wikipedia

³ Philippe Flajolet, Étienne Parizot, « Qu'est ce qu'un algorithme ? », interstices.fr, 2004.

L'instruction **Lire** peut donner la main à l'utilisateur pour entrer plusieurs valeurs. Les deux écritures suivantes sont correctes :

Lire(a) Lire(b)	Lire(a,b)
--------------------	-----------

L'instruction d'affectation a toujours une variable dans sa partie gauche et peut avoir dans sa partie droite, soit une constante ($A \leftarrow 3.14$), soit une variable qui a une valeur ($A \leftarrow B$), soit une expression évaluable ($A \leftarrow B*2+3$). Des écritures comme : $3.14 \rightarrow A$, $A + 2 \leftarrow B$ et $A \leftarrow B \leftarrow 5$ sont incorrectes syntaxiquement.

L'instruction **Ecrire** affiche soit, des valeurs de variables : `Ecrire(A)`, soit des messages : `Ecrire("Bonjour")`, soit une combinaison de messages et variables : `Ecrire("l'aire du cercle est ", A, " cm2")`.

Par convention, l'instruction **Ecrire** affiche et passe à la ligne.

2.1.6. Algorithme vs Programme

L'objectif derrière l'écriture d'un algorithme, est de le transformer (le traduire) en programme exécutable sur machine.

Le **pseudo code** algorithmique permet d'écrire des algorithmes à la main afin de se focaliser sur les idées et le raisonnement, il n'est pas censé être exécuté directement sur machine, sauf après l'avoir transformé (traduit) en **code** écrit dans un langage de programmation comme le langage C, Visual Basic, Java, Python ...

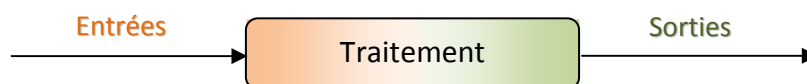
Donc c'est l'algorithmique qui a été créée au service de la programmation et non l'inverse.

Algorithme	Programme
Pseudo code	Code
N'est pas directement exécutable sur machine ⁴	Exécutable directement sur machine
Il a été créé au service de la programmation	

2.1.7. Utilisateur vs Programmeur (Développeur)

Il faut distinguer les deux rôles : utilisateur et programmeur. Un programmeur (ou développeur) conçoit des programmes destinés à être utilisés par des utilisateurs.

Quand on écrit des algorithmes, on joue le rôle de programmeur qui écrit un algorithme afin de le transformer (traduire) par la suite à un programme exécutable. L'utilisateur quand à lui, ne voit pas la suite d'instructions qu'un programmeur a écrit, mais voit uniquement les entrées (que le programmeur lui demande de donner) et les sorties que le programmeur lui affiche. Pour lui, ce qui se passe entre les entrées et les sorties est une boîte noire.



Le programmeur doit parfois se mettre à la place de l'utilisateur pour tester son programme lui-même, et joue ainsi les deux rôles.

⁴ Le cas d'AlgBox qui dans l'apparence permet d'exécuter des algorithmes, n'est dans la réalité qu'un outil pédagogique de simulation d'exécution d'un algorithme sur machine.

2.1.8. Conventions syntaxiques de base

Le pseudo code est purement conventionnel⁵, ce qui fait qu'on a la possibilité de définir les conventions qu'on juge appropriées à notre guise. Toutefois, certaines conventions se sont imposées par la force des choses au fil des années, influencées par les langages de programmation de l'époque, comme Pascal.

On peut résumer les conventions syntaxiques de base qu'on va adopter tout au long de l'année comme suit :

- Il n'y a pas de distinction entre majuscules et minuscules dans les mots clés, ainsi qu'avec et sans accents. On peut écrire par exemple : DEBUT, Début ou debut.
- Il y a distinction entre majuscules et minuscules dans les variables. Les deux variables : Nom et NOM sont distinctes.
- Le nom de l'algorithme et des variables peut être composé de lettres et chiffres et doit obligatoirement commencer par une lettre, et ne doit pas contenir de lettres de ponctuation, l'espace en particulier. On peut écrire comme nom d'algorithme ex_01 et non pas ex 01.
- Les blancs et tabulations avant les instructions ne dérangent en rien, ainsi qu'avant et après les variables, les virgules et les parenthèses. Les sauts de lignes en plus ne dérangent en rien également.
- Les points-virgules à la fin des instructions ne sont pas obligatoires, avec une préférence pour les omettre.
- Les parenthèses sont obligatoires dans les deux instructions **Lire** et **Ecrire**. On écrit par exemple **Lire(a)** et non pas Lire a.
- Les messages écrits dans l'instruction **Ecrire**, doivent être délimités par " (double cotes) des deux cotés, comme : Ecrire("Bonjour").
- Les types de variables sont obligatoirement en singulier. On écrit : entier, et non pas entiers.
- On peut mettre plusieurs instructions sur une même ligne (mais on préfère mettre une par ligne).
- Les mots clés basiques adoptés sont :
 - Algo ou Algorithme (avec une préférence pour la première forme)
 - Variables ou Variable (avec une préférence pour la première forme)
 - Début
 - Fin
 - Lire
 - Ecrire
 - Entier
 - Réel
 - Car ou Caractère (avec une préférence pour la première forme)
 - chCar ou Chaine de Caractères (avec une préférence pour la première forme)
 - ← comme symbole d'affectation
 - * comme opérateur de multiplication au lieu de ×. (+ :pour l'addition, – :pour la soustraction, / :pour la division, il n'y a pas de symbole équivalent à l'opération mathématique d'élévation à la puissance)

2.1.9. Déroulement manuel d'un algorithme

Le processus de déroulement manuel d'un algorithme permet de suivre à la main, le résultat d'exécution, instruction par instruction. Ce qui donnera une idée de l'exécution sur machine, du même algorithme après l'avoir traduit en programme.

⁵ « Algorithmique et programmation pour non-matheux », Christophe Darmangeat, 2013, Université Paris 7, <http://pise.info/algo/introduction.htm>

Pour pouvoir dérouler (exécuter) un algorithme à la main, on dresse un tableau, avec un nombre de colonnes égal aux nombre de variables + 1 (la dernière colonne est réservée pour représenter ce qui est affiché sur l'écran), et un nombre de lignes égal aux nombre d'instructions + 1 (une instruction par ligne + la ligne comportant les noms de variables).

Exemple

Algorithme	Déroulement de l'algorithme avec la valeur 2.5 :		
Algo Aire Variables R, A : Réel Début Lire(R) $A \leftarrow 3.14 * R * R$ Ecrire(A) Fin	R	A	Ecran
	2.5		
		19.625	
			19.625

On peut faire correspondre plusieurs instructions à une seule ligne du tableau d'exécution.

L'ordre de déroulement des instructions est important, on les déroule instruction par instruction du début à la fin.

2.1.10. Questions de compréhension

1. L'algorithmique c'est : <input type="checkbox"/> A. du code <input type="checkbox"/> B. du pseudo code. <input type="checkbox"/> C. un mélange entre du code et du pseudo code. <input type="checkbox"/> D. du texte simple.	2. <input type="checkbox"/> A. l'algorithmique a été créée au service de la programmation. <input type="checkbox"/> B. la programmation a été créée au service de l'algorithmique. <input type="checkbox"/> C. l'algorithmique est la programmation sont indépendants.	3. Dans le langage C, les points virgules sont : <input type="checkbox"/> A. absents. <input type="checkbox"/> B. facultatifs. <input type="checkbox"/> C. obligatoires.
4. Chaque ligne du corps de l'algorithme s'appelle : <input type="checkbox"/> A. Ordre. <input type="checkbox"/> B. Opération. <input type="checkbox"/> C. Instruction. <input type="checkbox"/> D. ligne.	5. La première ligne de l'algorithme s'appelle : <input type="checkbox"/> A. tête. <input type="checkbox"/> B. entête. <input type="checkbox"/> C. introduction. <input type="checkbox"/> D. début.	6. Après la première ligne de l'algorithme vient le bloc : <input type="checkbox"/> A. Définition. <input type="checkbox"/> B. Déclaration. <input type="checkbox"/> C. Typage. <input type="checkbox"/> D. En-tête.
7. Dans nos conventions syntaxiques algorithmiques, les points virgules sont : <input type="checkbox"/> A. absents. <input type="checkbox"/> B. facultatifs. <input type="checkbox"/> C. obligatoires.	8. L'algorithme est censé être : <input type="checkbox"/> A. directement exécutable sur machine. <input type="checkbox"/> B. non directement exécutable sur machine. <input type="checkbox"/> C. exécutable uniquement après transformation en code.	9. L'instruction qui donne la main à l'utilisateur pour entrer une valeur est: <input type="checkbox"/> A. Afficher. <input type="checkbox"/> B. Ecrire. <input type="checkbox"/> C. Lire. <input type="checkbox"/> D. Entrer.
10. Nous écrivons l'instruction d'affectation comme suit : <input type="checkbox"/> A. = <input type="checkbox"/> B. == <input type="checkbox"/> C. ← <input type="checkbox"/> D. reçoit	11. L'instruction Ecrire permet : <input type="checkbox"/> A. d'entrer des données. <input type="checkbox"/> B. d'afficher des résultats sur écran <input type="checkbox"/> C. d'afficher des résultats imprimés sur papier. <input type="checkbox"/> D. d'imprimer des résultats sur écran.	12. L'ordre des instructions d'un algorithme est : <input type="checkbox"/> A. important <input type="checkbox"/> B. n'est pas important

2.1.11 Exercices

Exercice 01 :

L'algorithme de la 1^{ère} case est écrit syntaxiquement correct. Repérez les erreurs syntaxiques dans chacune des écritures suivantes (2, 3, 4, 5, 6, 7 et 8).

1	2	3	4	5	6	7	8
Algo Exo2 Variables a,b,c:Entier Début Lire(a) $b \leftarrow 2$ $c \leftarrow a * b$ Ecrire(c) Fin	Algo Exo2 Variables a,b,c:entier Début lir(a) $b \leftarrow 2$ $c \leftarrow a * b$ Ecrire c Fin	Algo Exo2 Variables a,b,c:Entiers Début Lire (a) $b \leftarrow 2$ $c \leftarrow a * _b$ ecrire(c) Fin	Algo Exo2 ; Variables; a,b,c Entier; Début ; LIRE(a) ; $b \leftarrow 2$; $c \leftarrow a * b$; Ecrire(c) ; Fin.	Algo Exo2 Variables a b c:Entier Début Lire(a) $b \rightarrow 2$ $c \leftarrow a * b$ Ecri(c) Fin	Algo Exo2 Variables c,a,b :Entier Début Lire(a) $b = 2$ $c \leftarrow a \times b$ Ecrire(c) Fin	Algo Exo2 Variables A,b,c:Entier Début Lire(A) $B \leftarrow 2$ $c \leftarrow a * b$ écrire(c) Fin	Algo Exo2 Variables af,b,c: Entier Début Lire(af) $b \leftarrow 2$ $c \leftarrow af * b$ Ecrire(e) Fin

Exercice 02 :

a- Exécuter l'algorithme suivant pour les valeurs PI=150 et R=30.

Algo Exo2

Variables

PI, R, PaP : Réel

Début

Ecrire ("Donnez SVP le prix initial et la remise")

Lire (PI,R)

$PaP \leftarrow PI - PI * R / 100$

Ecrire ("Le prix à payer =", PaP)

Fin

b- Que fait cet algorithme (quel est son rôle) ?

c- Qu'est ce qui s'affiche à l'écran pour chacun des cas suivants :

$PaP \leftarrow 105$ Ecrire("Le prix à payer =", PaP)	$PaP \leftarrow 105$ Ecrire("Le prix à payer = PaP")	$PaP \leftarrow 105$ Ecrire("Le prix à payer = " PaP)
$PI \leftarrow 150$ $PaP \leftarrow 105$ Ecrire("Le prix initial = PI et le prix à payer = PaP")	$PI \leftarrow 150$ $PaP \leftarrow 105$ Ecrire("Le prix initial =", PI, " et le prix à payer =", PaP)	
$PI \leftarrow 150$ $PaP \leftarrow 105$ Ecrire("Le prix initial = PI et le prix à payer =", PaP)	$PI \leftarrow 150$ $PaP \leftarrow 105$ Ecrire("Le prix initial =", "PI et le prix à payer =", PaP)	

2.2. Algorithmes équivalents

Soit les deux algorithmes suivants qui permettent de calculer l'aire d'un cercle à partir de son rayon saisi par l'utilisateur.

Algo Aire1

Variables

R, A : Réel

Début

Lire (R)

$A \leftarrow 3.14 * R * R$

Ecrire (A)

Fin

Algo Aire2

Variables

R, A, Pi : Réel

Début

Lire (R)

$Pi \leftarrow 3.14$

$A \leftarrow Pi * R * R$

Ecrire (A)

Fin

Les deux algorithmes sont corrects, et si on les exécute (après les avoir traduits en programmes), l'utilisateur constatera qu'ils font exactement la même chose ; lui demander de saisir la valeur du rayon, et afficher l'aire du cercle correspondant. Ces deux algorithmes sont dits **équivalents**.

- On dit que deux algorithmes sont équivalents quand ils ont le même comportement du point de vue utilisateur en ce qui concerne les entrées et les sorties correspondantes.
- On dit que deux algorithmes sont équivalents quand ils donnent toujours les mêmes résultats par rapport aux mêmes jeux d'entrée.

2.2.1. Algorithme optimal (meilleur)

La question qui découle directement de la notion d'équivalence des algorithmes est la suivante :

- Entre deux (ou plusieurs) algorithmes équivalents, quel est le meilleur ?

Pour répondre à cette question, il y a plusieurs critères. Nous allons à ce niveau de 1^{ère} année, nous focaliser sur deux points : le temps d'exécution et la consommation d'espace mémoire.

Pour le premier point, réduire le temps d'exécution revient à réduire le nombre d'instructions et/ou d'opérations. Pour le deuxième point, réduire l'espace mémoire revient à réduire le nombre de variables utilisées.

Dans l'exemple précédent, l'algorithme Aire1 est meilleur que l'algorithme Aire2, puisqu'il a un nombre d'instructions et de variables moindre.

Il n'est pas toujours possible d'avoir un algorithme optimal en temps d'exécution et en espace mémoire en même temps. Dans ce cas, on essaye de trouver un compromis entre les deux critères.

Dans notre niveau de 1^{er} semestre de 1^{ère} année, on demande généralement aux étudiants d'écrire des algorithmes **corrects** et non pas nécessairement optimaux.

2.2.2. Messages indicatifs et explicatifs

Pour éviter que l'utilisateur ne soit perdu et ne sache quoi saisir quand il verra simplement le curseur entrain de clignoter, on ajoute généralement des messages indicatifs et explicatifs (libellés) pour rendre l'exécution plus compréhensible par l'utilisateur.

On peut par exemple modifier l'algorithme Aire1 comme suit :

Algo Aire1

Variables

R, A : Réel

Début

Ecrire ("Entrez SVP le rayon d'un cercle en Cm")

Lire (R)

$A \leftarrow 3.14 * R * R$

Ecrire ("l'aire de ce cercle est : ", A, " Cm2")

Fin

Les messages indicatifs et explicatifs n'entrent pas en compte dans la recherche d'optimalité des algorithmes.

2.2.3. Questions de compréhension

1. Deux algorithmes sont dit équivalents : <input type="checkbox"/> A. quand ils ont uniquement les mêmes entrées. <input type="checkbox"/> B. quand ils ont toujours les mêmes résultats par rapport aux mêmes jeux d'entrée. <input type="checkbox"/> C. quand ils ont le même jeu d'instructions.	2. On juge qu'un algorithme est meilleur qu'un autre sur deux points : <input type="checkbox"/> A. la lisibilité et le confort. <input type="checkbox"/> B. la vitesse et le confort du programmeur. <input type="checkbox"/> C. le temps d'exécution et l'espace mémoire.	3. Quand X= 12, l'instruction écrire("la moyenne est:", X), affiche <input type="checkbox"/> A. la moyenne est: 12 <input type="checkbox"/> B. la moyenne est:12 <input type="checkbox"/> C. la moyenne est:X
	4. Quand X= 12, l'instruction écrire("la moyenne est:", X, "sur 20"), affiche : <input type="checkbox"/> A. la moyenne est: Xsur 20 <input type="checkbox"/> B. la moyenne est: 12 sur 20 <input type="checkbox"/> C. la moyenne est:12sur 20	

2.2.4. Exercices

Exercice 01 :

Soit le traitement suivant :

- Donner la main à l'utilisateur pour **entrer** un nombre entier.
- Ajouter 5 à ce nombre.
- Multiplier le résultat par 3.
- Afficher le résultat final.

Ecrire six (6) variantes d'algorithmes qui réalisent le traitement précédent et **exécuter (dérouler)** à chaque fois avec la valeur 1 :

- Avec 4 **instructions** et 3 **variables** ; une seule instruction pour chacune des étapes et une variable distincte pour chacune des trois premières instructions.
- Avec 4 instructions et 2 variables.
- Avec 4 instructions et 1 variable.
- Avec 3 instructions et 2 variables.
- Avec 3 instructions et 1 variable.
- Avec 2 instructions et 1 variable.

Exercice 02 :

On veut **permuter** les valeurs de deux variables entières A et B.

1- Exécuter les deux algorithmes suivants avec les valeurs (6, 17) :

Algo Exo9_2_A

Variables

A, B : Entier

Début

Lire(A, B)

$A \leftarrow B$

$B \leftarrow A$

Ecrire("Les nouvelles valeurs de A et B sont ", A, " ", B)

Fin

Algo Exo9_2_B

Variables

A, B : Entier

Début

Lire(A, B)

$B \leftarrow A$

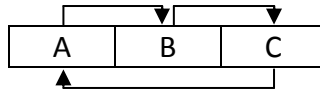
$A \leftarrow B$

Ecrire("Les nouvelles valeurs de A et B sont ", A, B)

Fin

2- Réécrire l'algorithme pour permettre de permuter les deux variables de façon correcte, et vérifiez-le en l'exécutant avec les mêmes valeurs de la question 1.

3- Ecrire l'algorithme qui permet de permuter les valeurs de 3 variables A, B et C, de façon circulaire comme suit :



4- Généraliser pour 4 et 5 variables.

2.3. Types de variables

Une variable est l'association d'un identifiant (un nom de variable) et d'une case mémoire pouvant contenir une donnée. Une variable peut être du type **Entier** comme l'exemple de la section 2.1.2, **Réel** comme l'exemple de la section 2.1.5, **caractère**, ou **chaîne de caractères** comme l'exemple suivant :

Algo Bienvenue

Variables

N, P : chCar

Début

Ecrire ("Quel est SVP votre nom et votre prénom ?")

Lire (N, P)

Ecrire ("Bonjour ", P, " ", N)

Fin

Dans cet exemple, on demande à l'utilisateur de donner son nom et son prénom et on affiche un message de bienvenue : **Bonjour prénom nom**. Si l'utilisateur saisit **Ayad** pour le nom et **Samir** pour le prénom, l'algorithme affichera : **Bonjour Samir Ayad**

La seule opération qu'on peut appliquer sur les chaînes de caractères pour le moment, est la **concaténation**, qu'on symbolise avec le caractère +.

L'algorithme précédent peut être réécrit comme suit :

Algo Bienvenue2

Variables

N, P, Ncomplet : chCar

Début

Ecrire ("Quel est SVP votre nom et votre prénom ?")

Lire (N, P)

Ncomplet \leftarrow P + " " + N

Ecrire ("Bonjour ", Ncomplet)

Fin

Pour le type caractère, voici un exemple :

Algo Bienvenue3

Variables

LN, LP : Car

Initiales : chCar

Début

Ecrire ("Donnez SVP la première lettre de votre nom et de votre prénom")

Lire (LN, LP)

Initiales \leftarrow LP + '.' + LN

Ecrire ("Bonjour ", Initiales)

Fin

Dans cet exemple : si l'utilisateur entre LN = **A** et LP = **S**, l'algorithme affiche : **Bonjour A.S**

Remarquez qu'on délimite un caractère (comme le point dans cet exemple) par simples cotes et non pas doubles cotes comme pour les chaînes de caractères.

2.3.1. Exercice

Ecrire un algorithme qui demande à l'utilisateur d'entrer son nom, son prénom, son âge et son adresse. Et affiche les informations entrées comme suit :

+++++

Bonjour Ayad Halim

Vous avez : 18 ans

Vous habitez : Cité Benboulaïd Constantine

2.3. Traitement conditionnel (Alternatif)

Soit l'algorithme suivant qui se propose de résoudre une équation du 1^{er} degré comme : $2X - 6 = 0$

Algo equation_1er_degre

Variables

a,b,X : Réel

Début

Lire(a, b)

$X \leftarrow -b/a$

Ecrire(X)

Fin

Cet algorithme fonctionne très bien sauf dans le cas où l'utilisateur entre un facteur **a**, nul. Dans ce cas, l'exécution se bloque au niveau de l'instruction $X \leftarrow -b/a$, et ne peut aller au-delà.

Pour remédier à ce problème, on fait recours à un outil algorithmique nouveau, qui est le traitement conditionnel :

Algo equation_1er_degre_B

Variables

a,b,X : Réel

Début

Lire(a, b)

Si $a \neq 0$:

$X \leftarrow -b/a$

Ecrire(X)

Fsi

Fin

Dans ce traitement conditionnel, l'algorithme ne procède au calcul que si le facteur **a** est non nul. Ce qui évitera le cas du blocage si $a = 0$.

L'inconvénient de ce nouvel algorithme est qu'il n'affiche rien à l'utilisateur si $a = 0$. Pour remédier à cela, on peut rajouter un bloc **Sinon** au bloc **Si** comme suit :

Algo equation_1er_degre_C

Variables

a,b,X : Réel

Début

Lire(a, b)

Si $a \neq 0$:

$X \leftarrow -b/a$

Ecrire(X)

Sinon

Ecrire("Erreur : Entrez une valeur du 1er facteur différente du zéro")

Fsi

Fin

Le fait d'écrire un message d'erreur dans le cas où $a = 0$ est un choix du programmeur. On aurait pu le traiter comme cas particulier, parce que du point de vue mathématique :

- Il y a une infinité de solutions si $a = 0$ et $b = 0$.
- Il n'y a pas de solution si $a = 0$ et $b \neq 0$ (on lit : b différent de zéro).

On pourra donc faire :

Algo equation_1er_degre_D

Variables

a,b,X : Réel

Début

Lire(a, b)

Si $a \neq 0$:

$X \leftarrow -b/a$

Ecrire(X)

Sinon

Si $b = 0$:

Ecrire("infinité de solutions")

Sinon

Ecrire("Pas de solution")

Fsi

Fsi

Fin

On appelle les deux traitements alternatifs précédents, l'un à l'intérieur de l'autre, des traitements alternatifs imbriqués.

On a deux formes possibles du traitement conditionnel, qu'on peut imbriquer ou non :

1 ^{ère} forme	2 ^{ème} forme	Exemple de traitements conditionnels séquentiels (séparés)	Exemple de traitements alternatifs imbriqués
Si Condition : Suite d'instructions Fsi	Si Condition : Suite1 d'instructions Sinon Suite2 d'instructions Fsi	Si Condition1 : Suite1 d'instructions Fsi Si Condition2 : Suite2 d'instructions Sinon Suite3 d'instructions Fsi Si Condition3 : Suite4 d'instructions Sinon Suite5 d'instructions Fsi	Si Condition1 : Si Condition2 : Suite1 d'instructions Sinon Suite2 d'instructions Fsi Sinon Suite3 d'instructions Fsi

La condition des traitements alternatifs peut être composée de plusieurs termes logiques reliés par des opérateurs logiques (Non, Et, Ou). Prenons comme exemple, la condition qui permet de vérifier si une année est bissextile ou non. Selon les calculs astronomiques, une année est bissextile si elle est :

- Divisible par 4 et non divisible par 100.
- Ou
- Divisible par 400.

Cette condition sera exprimée algorithmiquement comme suit :

Si $(A \text{ Mod } 4 = 0 \text{ et } A \text{ Mod } 100 \neq 0) \text{ ou } A \text{ Mod } 400 = 0$:

...

Il y a un ordre de priorité entre les opérateurs logiques comme pour les opérateurs arithmétiques. Le **Non** est prioritaire par rapport au **Et** logique, lequel est prioritaire par rapport au **Ou** logique.

2.3.1. Conventions syntaxiques des conditions

Dans les conventions syntaxiques algorithmiques qu'on va adopter tout au long de l'année :

- On peut écrire l'inégalité inférieur ou égal soit : \leq ou \leq , mais pas $=<$ (De même pour supérieur ou égal, soit \geq ou \geq , mais pas $=>$).
- Dans une expression logique, les parenthèses ne sont pas obligatoires. Un ordre de priorité entre les opérateurs logiques permet d'enlever l'ambiguïté.
- L'opérateur égal s'écrit dans nos conventions syntaxiques $=$ est non pas $==$ comme en langage C.
- L'opérateur **ET** logique s'écrit dans nos conventions syntaxiques **ET** et non pas $\&\&$ comme en langage C (Idem pour le **OU** et le **NON**).
- La condition du **Si** peut être écrite sur plusieurs lignes.

2.3.2. Equivalent du traitement conditionnel dans les langages de programmation

Les conventions syntaxiques algorithmiques du traitement conditionnel ont un équivalent qui diffère légèrement d'un langage de programmation à un autre.

Algorithmique	Vb	C	Python
Si N < 10 : N = N + 1 Sinon N = N - 1 FSi	If N < 10 Then N = N + 1 Else N = N - 1 End If	If(N < 10) { N = N + 1 } else { N = N + 1 }	if N < 5: a = a + 1 else: a = a - 1

2.3.3. Traitements conditionnels séquentiels vs imbriqués

Soit l'exemple suivant : on voudra écrire un algorithme qui permet à l'utilisateur d'entrer un nombre entre 1 et 12 qui représente le numéro du mois, et affiche dans quel trimestre il appartient. Ex : si l'utilisateur tape 5, on veut afficher **2eme trimestre**.

On a principalement quatre (4) cas pour quatre (4) trimestres et un dernier cas pour les valeurs non valides (<1 ou >12). On va commencer par écrire un algorithme avec cinq (5) traitements conditionnels séquentiels (séparés) :

Algo Trimestre1

Variables

M : Entier

Début

Lire(M)

Si M < 1 ou M > 12 :

 Ecrire("Données non valides")

Fsi

Si M \geq 1 et M \leq 3 :

 Ecrire("1er trimestre")

Fsi

Si M \geq 4 et M \leq 6 :

 Ecrire("2eme trimestre")

Fsi

Si M \geq 7 et M \leq 9 :

 Ecrire("3eme trimestre")

Fsi

Si M \geq 10 et M \leq 12 :

 Ecrire("4eme trimestre")

Fsi

Fin

L'algorithme précédant couvre tous les cas possibles. L'ordre des différents cas n'est pas important, on peut commencer par celui qu'on veut, puisque on va les tester un par un.

Réécrivons maintenant le même algorithme avec des traitements alternatifs imbriqués

Algo Trimestre2

Variables

M : Entier

Début

Lire(M)

Si $M \leq 0$ ou $M > 12$:

Ecrire("Données non valides")

Sinon

Si $M \leq 3$:

Ecrire("1er trimestre")

Sinon

Si $M \leq 6$:

Ecrire("2eme trimestre")

Sinon

Si $M \leq 9$:

Ecrire("3eme trimestre")

Sinon

Ecrire("4eme trimestre")

Fsi

Fsi

Fsi

Fsi

Fin

La première remarque dans le nouvel algorithme est qu'on n'a pas écrit le dernier test, puisque dans le cas des traitements alternatifs, le dernier **Sinon** représente le cas restant.

La deuxième remarque est que les conditions des traitements alternatifs ont été raccourcies puisque le fait d'être dans le **Sinon**, implique déjà la négation de la condition du **Si**.

La troisième remarque est que le dernier algorithme est meilleur en ce qui concerne son temps d'exécution puisque dans le cas de la vérification de l'un des premiers tests, il n'a pas à vérifier les autres, chose que le premier algorithme ne fait pas.

Pour faire la similitude avec un cas concret, on prend l'exemple de la recherche sur feuilles de présence, du numéro de groupe d'un étudiant dont on connaît le nom, prénom et numéro de carte et dont on est sûr de l'appartenance à un et un seul groupe d'une promotion donnée. Le deuxième algorithme est similaire au cas de l'arrêt de la recherche une fois l'étudiant trouvé dans l'un des groupes. Par contre le premier algorithme, continue la recherche comme même.

Dans le cas précis de ce dernier exemple, les deux algorithmes sont corrects avec une préférence pour les traitements alternatifs imbriqués qui sont meilleurs du point de vue temps d'exécution.

Cependant, ce n'est pas toujours le cas, les traitements conditionnels séquentiels séparés peuvent s'imposer pour d'autres types de problèmes, notamment lorsque les conditions des différents traitements sont indépendants, comme pour le cas classique de l'extraction du maximum d'entre plusieurs valeurs.

2.3.4. Traitement à choix multiple

L'algorithmique (et les langages de programmations derrière) offre un outil qui simplifie la tâche du programmeur lorsqu'il a à écrire plusieurs traitements conditionnels portants sur la même expression de variables.

Le dernier exemple peut être réécrit comme suit :

Algo Trimestre3

Variables

M : Entier

Début

Lire(M)

Selon M

1 à 3 : Ecrire("1er trimestre")

4 à 6 : Ecrire("2eme trimestre")

7 à 9 : Ecrire("3eme trimestre")

10, 11, 12 : Ecrire("4eme trimestre")

Autre : Ecrire("Données non valides")

FSelon

Fin

Faisant partie du code, qui est transparent pour l'utilisateur, cet outil ne va faciliter que la tâche du programmeur.

Cet outil ne peut remplacer les traitements alternatifs que si d'une part, leurs conditions portent sur la même expression de variables, et d'autre part, que les valeurs attendues appartiennent à des ensembles discrets finis, et non pas des intervalles continus.

Les traitements de l'exemple suivant ne peuvent être remplacés par **Selon**, puisque les intervalles sont continus :

Algo Mention

Variables

Moy : Réel

Début

Lire(Moy)

Si $M < 0$ ou $M > 20$:

Ecrire("Données non valides")

Sinon

Si $M \leq 10$:

Ecrire("Ajournée")

Sinon

Si $M \leq 12$:

Ecrire("Passable")

Sinon

Si $M \leq 14$:

Ecrire("Bien")

Sinon

Ecrire("Très bien")

Fsi

Fsi

Fsi

Fin

2.3.5. Exercices

Exercice 01:

L'algorithme de la 1^{ère} case est écrit syntaxiquement correct.

a- Repérez les erreurs syntaxiques dans chacune des écritures suivantes (de 2 à 15), s'il y en a.

1	2	3	4	5
Algo Exo2_3_1 Variables N : Entier Début Lire(N) Si N < 10 : Ecrire("chiffre") Sinon Ecrire("nombre") Fsi Fin	Algo Exo2_3_2 Variables N : Entier Début Lire(N) Si N < 10 Ecrire("chiffre") Sinon Ecrire("nombre") Fsi Fin	Algo Exo2_3_3 Variables N : Entier Début Lire(N) Si N < 10 : Ecrire("chiffre") Sinon Ecrire('nombre ') Finsi Fin	Algo Exo2_3_4 Variables N : Entier Début Lire(N) Si N < 10 : Ecrire("chiffre") Sinon Ecrire("nombre") Fsi Fin	Algo Exo2_3_5 Variables N : Entier Début Lire(N) Si N < 10 : Ecrire("chiffre") Sinon Ecrire("nombre") Fsi Fin
6	7	8	9	10
Algo Exo2_3_6 Variables N : Entier Début Lire(N) Si N < 10 : Ecrire("chiffre") Sinon Ecrire("nombre") Fin	Algo Exo2_3_7 Variables N : Entier Début Lire(N) Si N < 10 : Ecrire("chiffre") Ecrire("nombre") Fsi Fin	Algo Exo2_3_8 Variables N : Entier Début Lire(N) Si (N < 10) : Ecrire("chiffre") Sinon Ecrire("nombre") Finsi Fin	Algo Exo2_3_9 Variables N : Entier Début Lire(N) Si N < 10 Alors Ecrire("chiffre") Sinon Ecrire("nombre") Fsi Fin	Algo Exo2_3_10 Variables N : Entier Début Lire(N) Si N < 10 : Sinon Ecrire("nombre") Fsi Fin
11	12	13	14	15
Algo Exo2_3_11 Variables N : Entier Début Lire(N) Si N < 10 : Ecrire("chiffre") Sinon Fsi Fin	Algo Exo2_3_12 Variables N : Entier Début Lire(N) Si N < 10 : Si N < 2 : Ecrire("binaire") Fsi Ecrire("chiffre") Sinon Ecrire("nombre") Fsi Fin	Algo Exo2_3_13 Variables N : Entier Début Lire(N) Si N < 10 : Si N < 2 : Ecrire("binaire") Ecrire("chiffre") Sinon Ecrire("nombre") Fsi Fsi Fin	Algo Exo2_3_14 Variables N : Entier Début Lire(N) Si (N < 10) et (N ≥ 0) : Ecrire("chiffre") Sinon Ecrire("nombre") Fsi Fin	Algo Exo2_3_15 Variables N : Entier Début Lire(N) Si 0 ≤ N < 10 : Ecrire("chiffre") Sinon Ecrire("nombre") Fsi Fin

Dans l'algorithme de la 1^{ère} case :

b- Que se passe-t-il si l'utilisateur entre une valeur négative ? Comment corriger cette anomalie ?

c- Que se passe-t-il si l'utilisateur entre une valeur réelle ?

d- Que se passe-t-il si l'utilisateur entre un caractère ou une chaîne de caractères ? Comment corriger cette anomalie ?

Exercice 02:

- a- Ecrire un algorithme qui demande deux valeurs réelles et affiche la plus grande d'entre elles. Si les deux valeurs sont égales, il ne doit pas le préciser.
- b- Ecrire un algorithme qui demande trois valeurs réelles et affiche la plus grande d'entre elles. S'il y a deux valeurs égales ou plus, il ne doit pas le préciser.
- c- Généraliser pour 4 et 5 valeurs.

Exercice 03:

- a) Ecrire un algorithme qui demande à l'utilisateur de saisir quatre (4) nombres entiers inférieurs à 100, et affiche le plus petit d'entre eux.
- b) Réécrire l'algorithme pour afficher le plus petit nombre impair parmi les quatre saisis.

Exemple : Dans le cas où l'utilisateur entre 25, 6, 17, 6, l'algorithme affiche :

- 6 pour la question 'a'.
- 17 pour la question 'b'.

Exercice 04:

- a) Ecrire un algorithme qui demande à l'utilisateur de saisir quatre (4) nombres entiers pairs positifs, et affiche le plus grand d'entre eux.
- b) Réécrivez l'algorithme pour afficher le plus grand nombre multiple de 6 parmi les nombres saisis.

Exemple : Dans le cas où l'utilisateur entre 12, 38, 24, 38, l'algorithme affiche :

- 38 pour la question 'a'.
- 24 pour la question 'b'.

2.4. Traitement répétitif (Les boucles)

L'algorithmique dispose d'un outil, qui est les boucles, permettant de réaliser des traitements répétitifs. Il y a principalement trois types de boucles :

2.4.1. La boucle **Pour**

Soit l'exemple d'algorithme suivant qui demande une valeur entière et la multiplie cinq fois avant de l'afficher :

Algo MultiplierN

Variables

N : Entier

Début

Lire(N)

$N \leftarrow N * 2$

$N \leftarrow N * 2$

$N \leftarrow N * 2$

$N \leftarrow N * 2$

$N \leftarrow N * 2$

Ecrire(N)

Fin

Si l'utilisateur entre la valeur **3** comme entrée, l'algorithme la multipliera cinq fois (6, 12, 24, 48, 96) et affichera **96**.

L'algorithmique dispose d'un outil plus simple pour écrire les traitements répétitif. L'exemple précédent peut être écrit plus simplement :

Algo MultiplierN

Variables

N, i : Entier

Début

Lire(N)

Pour i \leftarrow 1 à 5 :

$N \leftarrow N * 2$

FPour

Ecrire(N)

Fin

La boucle **Pour** permet donc, d'exécuter un bloc d'instructions, un certain nombre de fois. On appelle ce nombre de répétitions d'une boucle, nombre d'**itérations**.

Le nombre d'itérations est précisé par la valeur du début et de la fin de la boucle (1 et 5 dans l'exemple précédent).

Les trois boucles suivantes sont tout à fait équivalentes :

Pour i \leftarrow 1 à 5 : $N \leftarrow N * 2$ FPour	Pour i \leftarrow 0 à 4 : $N \leftarrow N * 2$ FPour	Pour i \leftarrow -3 à 1 : $N \leftarrow N * 2$ FPour
--	--	---

On appelle i, l'indice de la boucle qui représente le numéro d'itération.

Pour des raisons pédagogiques, on va toujours utiliser i (et par la suite j et k...) pour les indices des boucles Pour (sauf cas particulier), et garder le nom de variable Co pour la notion de compteur qu'on va voir dans la section 2.4.7.

La boucle **Pour** a une forme plus générale permettant une exécution répétitive avec un **Pas** :

Pour $i \leftarrow 1$ à 9 pas = 2: $N \leftarrow N * 2$ FPour	Permet de multiplier N cinq fois, pour $i = 1, 3, 5, 7$ et 9.
Pour $i \leftarrow 9$ à 1 pas = -2: $N \leftarrow N * 2$ FPour	Permet de multiplier N cinq fois, pour $i = 9, 7, 5, 3$ et 1.
Pour $i \leftarrow 5$ à 1 pas = -1: $N \leftarrow N * 2$ FPour	Permet de multiplier N cinq fois, pour $i = 5, 4, 3, 2$ et 1.

Quand on omet de mettre la valeur du **Pas**, elle est par défaut égale à **1**, comme dans les premiers exemples de cette section (2.4.1).

2.4.1.1 Equivalent de la boucle Pour dans les langages de programmation

Algorithmique	VB	C	Python
Pour $i \leftarrow 1$ à 9 pas = 2: $N \leftarrow N * 2$ FPour	For i = 1 To 9 Step 2 $N = N * 2$ Next	for(i=1;i<=9; i+=2) { $N = N * 2$; } }	for i in range(1,5,2): $N = N * 2$

2.4.2. La boucle TantQue

Soit l'exemple d'algorithme suivant, qui demande un nombre entier **N** et le multiplie jusqu'à ce que sa valeur dépasse 100. Ici on ne pourra pas utiliser la boucle **Pour** puisqu'on ne peut pas à priori, sans calcul mathématique particulier, connaître le nombre de répétitions.

Une autre boucle plus adaptée à ce type de traitement répétitif, est la boucle **TantQue** qu'on peut appliquer à l'énoncé précédent comme suit :

Algo MultiplierN_inferieur_100

Variables

N : Entier

Début

Lire(**N**)

TQ $N \leq 100$:

$N \leftarrow N * 2$

FTQ

Ecrire(**N**)

Fin

Si l'utilisateur donne à **N** la valeur 3, la boucle **TantQue** la multiplie jusqu'à dépassement de 100, en obtenant 6, puis 12, puis 24, puis 48, puis 96, et s'arrête sur 192 (qui est la première valeur supérieure à 100).

2.4.2.1 Equivalent de la boucle TantQue dans les langages de programmation

Algorithmique	VB	C	Python
TQ $N \leq 100$: $N \leftarrow N * 2$ FTQ	While N <= 100 $N = N * 2$ Wend	while(N <= 100) { $N = N * 2$ } }	while i <= 100: $N = N * 2$

2.4.3. La boucle Répéter

La boucle **Répéter** quant à elle, fait la même chose que la boucle **TantQue**, sauf qu'elle teste une condition à la fin de l'exécution de chaque itération.

L'exemple précédent peut être écrit avec la boucle **Répéter** comme suit :

Algo MultiplierN_inferieur_100

Variables

N : Entier

Début

Lire(N)

Rpt

N ← N * 2

Jsq N > 100

Ecrire(N)

Fin

On lit, « Répéter N reçoit N fois 2, jusqu'à N supérieur à 100 ».

Remarquez qu'en algorithmique, la condition de la boucle **Répéter** est la négation de la condition que nous avons écrit dans la boucle **TantQue**. Ceci est dû au fait, que pour la première boucle, la condition représente le critère d'arrêt des répétitions, tandis que pour la deuxième, elle représente la condition de poursuite des répétitions.

Dans ce dernier exemple, si l'utilisateur entre la valeur 3, la boucle **Répéter** la multiplie de façon répétitive : 6, 12, 24, 48, 96, et s'arrête sur 192. Les deux boucles **TantQue** et **Répéter** affiche dans ce cas, la même valeur. Mais si l'utilisateur entre la valeur 110 par exemple, la boucle **TantQue** de notre exemple n'effectue aucune multiplication et affiche 110, tandis que la dernière boucle **Répéter** exécute une fois et affiche 220.

Les deux boucles donc, ne sont pas tout à fait équivalente ; la différence entre **Répéter** et **TantQue** est que la première exécute son bloc d'instructions au moins une fois, alors que la boucle **TantQue** peut ne pas exécuter son bloc du tout (exécuter son bloc zéro fois).

La boucle Répéter est très adaptée par exemple pour la vérification de la saisie de l'utilisateur dans le traitement des cas d'erreur, comme dans l'exemple suivant :

Algo MultiplierN_inferieur_100

Variables

N : Entier

Début

Rpt

Ecrire("Entrez SVP un nombre entier non nul")

Lire(N)

Jsq N ≠ 0

Ecrire(1/N)

Fin

2.4.3.1 Equivalent de la boucle Répéter dans les langages de programmation

Algorithmique	VB	C	Python
Rpt N ← N * 2 Jsq N > 100	Do N = N * 2 Loop Until N > 100	Do { N = N * 2 } while (N <= 100) ;	-

2.4.4. Boucles imbriquées

Il est tout à fait possible d'imbriquer des boucles, les unes à l'intérieur des autres. Aucune restriction sur le nombre et le type des boucles à imbriquer n'est imposée, tout dépend du besoin de l'algorithme à écrire. On peut donc mettre à volonté, n'importe quelle boucle à l'intérieur de n'importe quelle autre, et refaire ce processus autant de fois que nécessaire.

L'exemple suivant demande sept nombres et multiplie chacun d'entre eux cinq fois :

Algo MultiplierN

Variables

N, i : Entier

Début

Pour i ← 1 à 7 :

Lire(N)

Pour i ← 1 à 5 :

N ← N * 2

FPour

Ecrire(N)

FPour

Fin

2.4.5. Passage entre les boucles

La boucle **TantQue** étant la plus forte en termes de puissance d'expression des problèmes algorithmiques, on peut toujours transformer de façon automatique n'importe quelle boucle **Pour** ou boucle **Répéter** en boucle **TantQue**. Mais on ne le fait jamais de façon abusive, puisqu'il faut utiliser chaque outil, là où il convient de le faire.

Transformation d'une boucle Pour en boucle TantQue	
Pour i ← 1 à 5 : Bloc d'instructions A FPour	i ← 1 TQ i ≤ 5 : Bloc d'instructions A N ← N + 1 FTQ

Transformation d'une boucle Répéter en boucle TantQue	
Rpt Bloc d'instructions B Jsq N > 15	Bloc d'instructions B TQ i ≤ 15 : Bloc d'instructions B FTQ

2.4.6. Choix du type de boucle

Quand on est face à un besoin algorithmique en traitement répétitif, on doit en premier, savoir choisir quelle boucle utiliser.

De façon générale, on utilise la boucle **Pour** quand on connaît le nombre d'itérations au moment de son utilisation, on fait appel à la boucle **Répéter** quand on ne connaît pas le nombre d'itérations et qu'on veut exécuter au moins une fois, et on garde la boucle **TantQue** pour tous les cas restants.

Donc on commence toujours par voir la possibilité d'utilisation de la boucle **Pour**. C'est la plus adaptée pour les cas où on connaît le nombre d'itérations. Elle a l'avantage d'être très simple à utiliser, à assimiler par le développeur, et ne présente pas de risque de boucle infinie par rapport à l'incréméntation, qui n'est pas explicite.

Si on ne connaît pas le nombre d'itérations et qu'on a plutôt une condition, on fait recours à l'un des deux autres types de boucles. Là encore, on ne doit pas remplacer abusivement **Répéter** par **TantQue** comme l'exemple suivant :

```
Lire(a)
TQ a = 0 :
  Lire(a)
FTQ
```

On utilisera plus simplement la boucle **Répéter**, qui est adaptée à ce cas précis de répétition d'au moins une fois :

```
Rpt
  Lire(a)
Jsq a ≠ 0
```

D'autre part, on ne doit jamais, dans ce niveau d'étude, bricoler la boucle **Pour** afin de permettre un comportement plus général (comme dans les exemples suivants). On ne doit jamais par exemple toucher à l'intérieur de la boucle aux valeurs du début et fin, pour sortir plutôt, comme dans le deuxième cas suivant, mais utiliser plutôt selon le cas, la boucle **Répéter** ou **TantQue** :

<pre>Pour i ← 2 à N : N ← N Div 2 FPour</pre>	<pre>Pour i ← 2 à N : A ← A + 10 Si A > 100 : i ← N + 1 Fsi FPour</pre>	<pre>Pour i ← 2 à N : A ← A + 10 i ← i + 2 FPour</pre>
---	--	--

2.4.7. Quelques mécanismes algorithmiques répétitifs fondamentaux

Il existe deux mécanismes algorithmiques répétitifs fondamentaux, le premier pour compter et le deuxième pour sommer ou multiplier de façon itérative.

L'algorithme suivant donne la main à l'utilisateur pour donner 100 nombres entiers, et affiche le nombre d'éléments pairs parmi les 100 saisis, leur somme et leur produit.

Algo Compter_somme_produit

Variables

N, i, Co, S, P : Entier

Début

Co ← 0

S ← 0

P ← 1

Pour i ← 1 à 100 :

Lire(N)

Si i mod 2 = 0 :

Co ← Co + 1

S ← S + i

P ← P * i

FPour

Ecrire(Co, " ", S, " ", P)

FPour

Fin

2.4.8. Exercices

Exercice 01:

Exécuter les algorithmes suivants avec la valeur 67 et 112 :

Algo Exo1_4_a Variables N : Entier Début Lire(N) TQ N < 100 : N ← N + 10 FTQ Ecrire(N) Fin	Algo Exo1_4_b Variables N : Entier Début Lire(N) Rpt N ← N + 10 Jsg N ≥ 100 Ecrire(N) Fin	Algo Exo1_4_c Variables N, i : Entier Début Lire(N) Pr i ← 1 à 4 : N ← N + 10 FPr Ecrire(N) Fin
---	--	--

Quels sont les algorithmes équivalents parmi les trois ci-dessus ?

Exercice 02: (Dans chacune des questions de l'exercice, n'utilisez que la boucle **pour**, avec la boucle **répéter** pour les cas d'erreurs)

a- Écrire un algorithme qui demande un nombre entier strictement positif **N**, et calcule $2 \times N$, en ajoutant **2**, **N** fois avec une boucle. L'algorithme doit afficher le résultat final uniquement.

b- Écrire un algorithme qui demande un nombre entier strictement positif **N**, et calcule et affiche 2^N .

c- Écrire un algorithme qui demande un nombre réel **B** et un nombre entier strictement positif **P**, et calcule et affiche B^P .

d- Écrire un algorithme qui demande un nombre réel **B** et un nombre entier **P**, et calcule et affiche B^P .

Exercice 03: (Dans chacune des 3 premières questions de l'exercice, n'utilisez que la boucle **pour**, avec la boucle **répéter** pour les cas d'erreurs)

a- Écrire un algorithme qui calcule et affiche la factorielle d'un nombre naturel **N** : $5! = 1 \times 2 \times 3 \times 4 \times 5$.

b- Écrire un algorithme qui calcule et affiche B^P , tel que B réel quelconque et P naturel.

c- Écrire un algorithme qui demande à l'utilisateur d'entrer **x**, et calcule et affiche e^x , par le développement de Taylor avec 10 fractions :

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots + \frac{x^9}{9!}$$

Exercice 04:

a- Écrire un algorithme qui demande à l'utilisateur de saisir un nombre naturel non nul **N**, et affiche tous ses diviseurs.

b- Écrire un algorithme qui demande un nombre naturel non nul **N**, et affiche le nombre de ses diviseurs.

c- Écrire un algorithme qui demande un nombre naturel **N**, et affiche s'il est premier ou pas. **Note** : un nombre est dit premier s'il est divisible par un (1) et lui-même uniquement. Par convention, un (1) n'est pas premier.

d- Écrire un algorithme qui calcule la somme des **Nb** premiers nombres premiers (**Nb** naturel non nul).