

<p>1^{ère} </p>	<h1>Algorithmique</h1>	 <p>Lycée Polyvalent PIERRE EMILE MARTIN</p>
<p>Cours / TD</p>	<p><u>Mots clé</u> Algorithme, algorithme, affectation, alternative, répétitive, sous-programme</p>	

Centre d'intérêt

CI4 : Gestion de l'information / Structures matérielles et Logicielles

Connaissances abordées

Implémentation d'un programme dans un « composant programmable »

Application

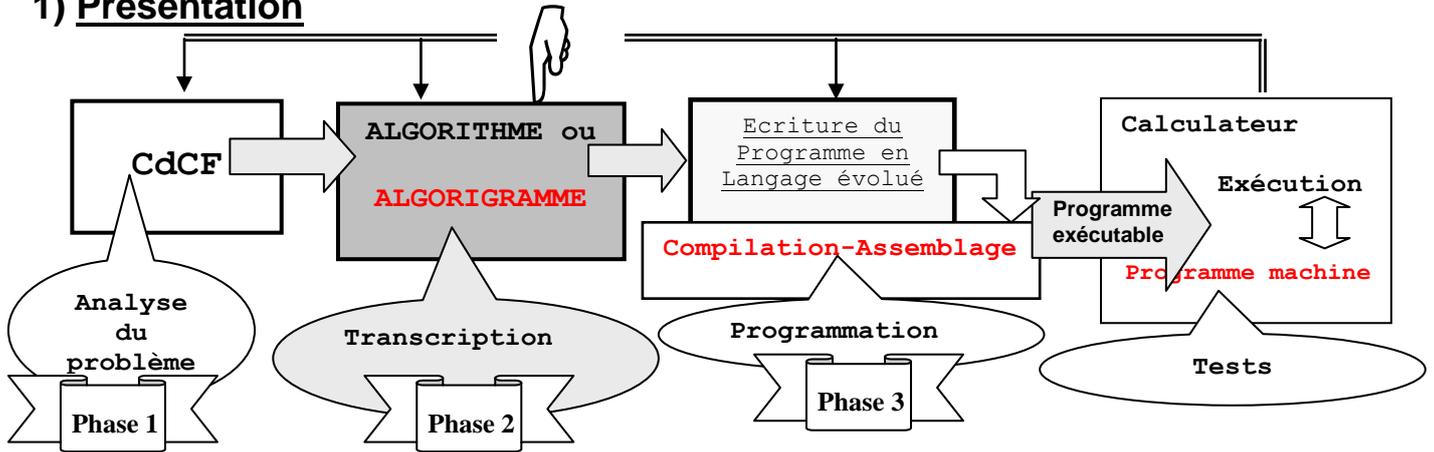
TP4, TP5, TP6 NXT + Microsoft Robotics Developer Studio (VPL). TP Arduino + Flowcode

```

Si (Taille(Match-List) > 1) Alors
C = ChoisirMeilleur(Match-List) ; {Choisit
dans Match-List}
Sinon
C = Premier(Match-List) ; {Choisit le clas
Match-List}
FinSi
    
```

Objectif du cours
Comprendre comment décrire un traitement devant être effectué par un calculateur.

1) Présentation



Le développement d'un programme nécessite trois phases.



Phase 1 Cahier Des Charges Fonctionnel (CdCF): Expression en français ou avec des outils de spécification du besoin.



Phase 2 ALGORITHMIQUE (Analyse structurée du problème)
Définition : « Un algorithme est une suite d'actions que devra effectuer un « calculateur » pour arriver, en un temps fini, à un résultat déterminé, à partir d'une situation donnée. La suite d'opérations sera composée d'actions élémentaires, ou **instructions** ».

Pour concevoir un algorithme trois étapes sont nécessaires :

- **La préparation du traitement** : recherche des données nécessaires à la résolution d'un problème.
- **Le traitement** : résolution pas à pas du problème après décomposition en plusieurs sous-ensembles si nécessaire.
- **L'édition** des résultats.

Phase 2

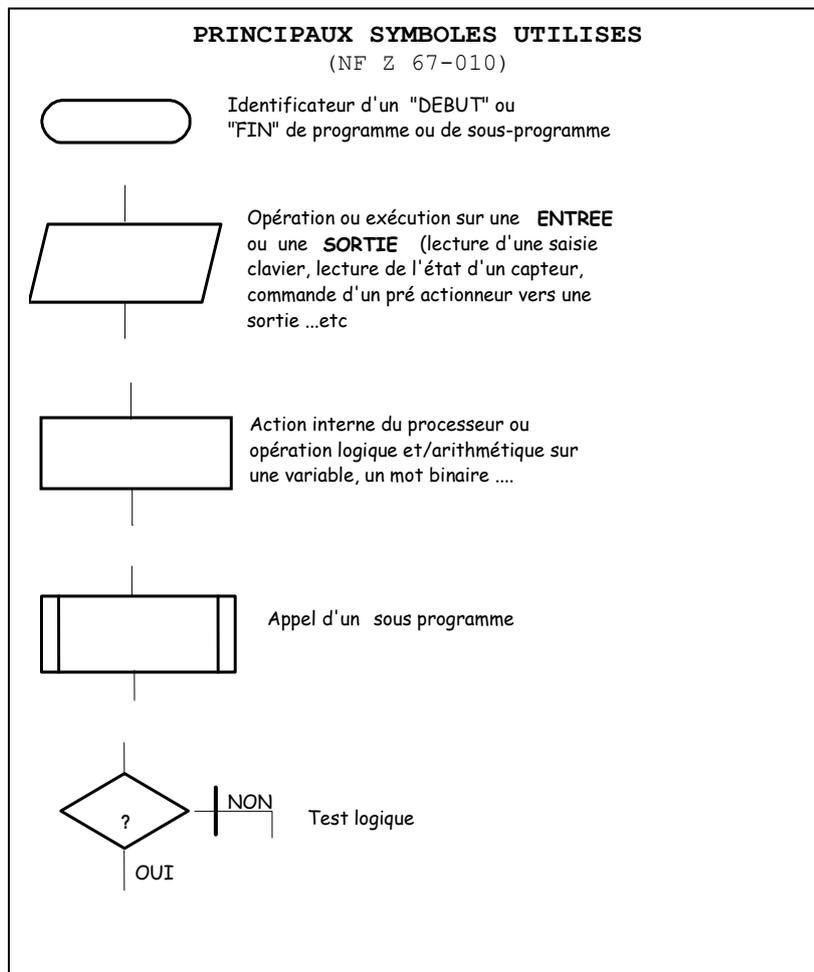
Modes d'expression d'un algorithme

Un algorithme peut être écrit :

- en **langage littéral (pseudo code)** par l'utilisation de **mots-clés** et de **délimiteurs**. On parle de langage algorithmique.

Exemples de mots-clés	Exemples de mots délimiteurs
Lire – Ecrire si.....alors... sinon tant que faire	Ils fixent les bornes des entrées et des sorties des structures algorithmiques : début - fin - fsi

- **graphiquement** (utilisation de symboles normalisés, on parle d'**algorithme**)



Phase 3

Traduction dans un langage « de programmation »

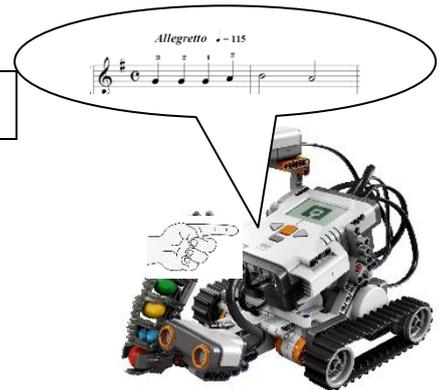
Les mots-clés ou les symboles graphiques sont remplacés par les mots appartenant à la syntaxe du langage utilisé. Cette dernière étape devrait être celle à laquelle le concepteur consacre **le moins de temps** (dans l'hypothèse ou les deux étapes précédentes ont été correctement développées !)

Exemple de traitement algorithmique

Enoncé : Permettre à une personne de jouer les premières notes de « Au clair de la lune » avec la brique du Lego NXT.

Phase 1 : Analyse du problème

Allegretto ♩ = 115

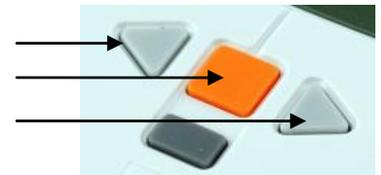


Phase 2 : Expression du problème en langage algorithmique

• 1 Préparation du traitement

52	Do4	C5	523,251
51	Si3	B4	493,883
50	La3#	A#4 ou Bb4	466,164
49	La3	A4	440
48	Sol3#	G#4 ou Ab4	415,305
47	Sol3	G4	391,995

Touche gauche => Sol
 Touche Enter => La
 Touche droite => Si



• 2 Traitement et édition du résultat

Algorithme AuClairDeLaLune

// Traitement : Jouer une note lors de l'appui sur une touche du clavier du NXT

//
 //

début

Lire (le clavier du NXT)

si (Tgauche appuyée) alors **Ecrire** (un Sol (F=391,995Hz))

sinon si (TEnter appuyée) alors **Ecrire** (un LA (F=440Hz))

sinon si (Tdroite appuyée) alors **Ecrire** (un SI (F=493,883Hz))

sinon rien

fsi

fsi

fsi

fin

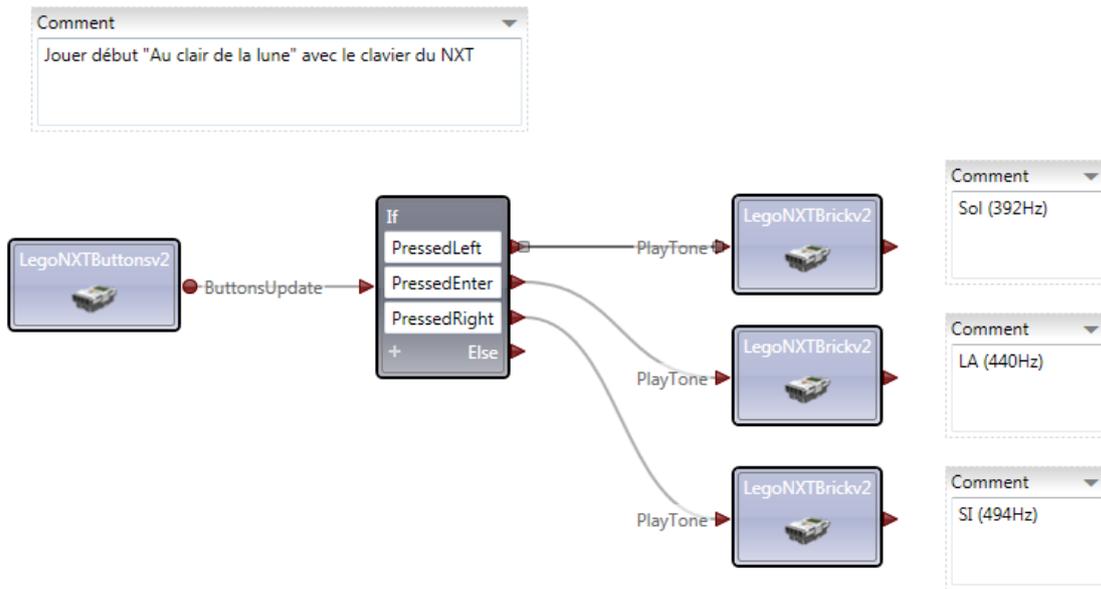
En-tête

Partie déclarative (vide ici)

Partie exécutive

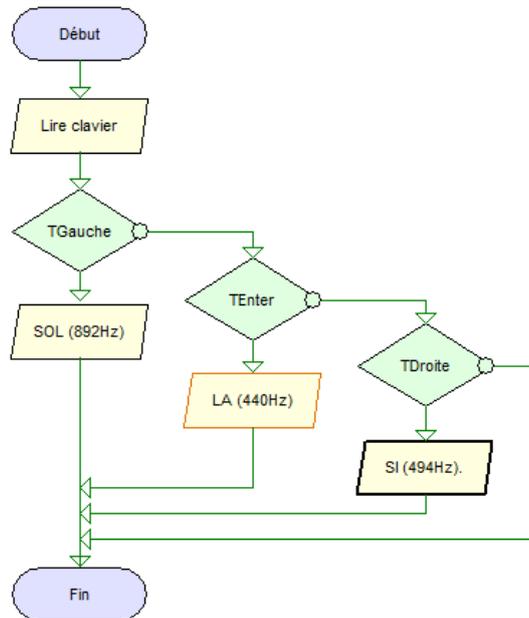
Le texte est indenté

Phase 3 : Traduction de l'algorithme dans un langage de programmation (implémentation).



Le problème précédent peut également être décrit par un **algorithme** (représentation graphique d'un algorithme).

Expression du problème précédent sous la forme d'un algorithme



Remarque : L'**algorithme** est bien adapté pour le codage dans un **langage de haut niveau** : (Ex : Java, C++, Basic, VPL...).



L'**algorithme** est plus adapté au codage dans un langage de **bas niveau** (Ex : assembleur) ou pour l'apprentissage de l'algorithmique (Ex : logiciel Flowcode)



2 Organisation d'un algorithme

Algorithmique

Raisonner pour concevoir

2.1 L'en tête

Dans cette partie le concepteur donne un **nom** à l'algorithme. Il définit le traitement effectué et les données auxquelles il se rapporte.

Exemple : **Algorithme** AuClairDeLaLune
// Traitement : Jouer une note lors de l'appui sur une touche du clavier du NXT

2.2 La partie déclarative

Dans cette partie, le concepteur décrit les différents « **objets** » que l'algorithme utilise.

2.2.1 Les constantes

Ce sont des « objets » constants dans tout l'algorithme.

Déclaration

Nom_Constante = valeur;



Exemple : Constantes Pi = 3,1416;

La déclaration de constantes symboliques permet de donner un nom à un objet constant dans tout l'algorithme et ensuite de faire référence à cet objet par son nom plutôt que par sa valeur.

2.2.2 Les variables

Ce sont des « objets » dont la valeur peut changer au cours de l'exécution de l'algorithme.

Déclaration

Nom_Variable : type;



Exemple : x, y : nombres réels;

Déclarer une variable consiste à définir son type.

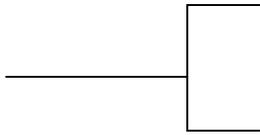
2.3 La partie exécutive

Elle est délimitée par les mots **début** et **fin**.

Algorithme	Algorigramme (NF Z 67-010)
<p>début</p> <p style="text-align: center;">....</p> <p style="text-align: center;">...</p> <p>fin.</p>	

2.4 Les commentaires

Des commentaires **doivent** être insérés dans le programme afin d'en faciliter la relecture.

Algorithme	Algorigramme (NF Z 67-010)
/* Commentaire sur plusieurs lignes */	
// Commentaire sur une seule ligne	

Exemple : // Traitement : Jouer une note lors de l'appui sur une touche du clavier du NXT

2.5 Le renvoi

Symbole utilisé deux fois pour assurer la continuité lorsqu'une partie de ligne de liaison n'est pas représentée dans un **algorigramme**.



3 Les « briques » d'un algorithme

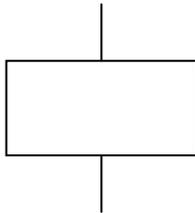
3.1 Les actions de base

- **L'affectation**

C'est l'**action essentielle** de l'algorithme Elle attribue une **valeur** (une constante ou le résultat d'un traitement) à une variable. On notera cette action par le symbole \leftarrow .

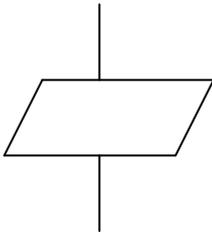
Exemples : $x \leftarrow 5$ $x \leftarrow x + 1$



Algorithme	Algorigramme (NF Z 67-010)
$Nom_Variable \leftarrow valeur$	

Remarque : L'affectation n'a de sens que si les deux objets de part et d'autre du signe \leftarrow sont de **même type**.

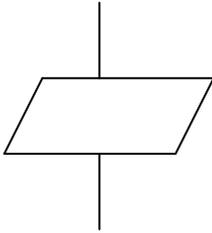
- **La saisie d'une valeur sur un terminal**

Algorithme	Algorigramme (NF Z 67-010)
Lire();	



Remarque : On pourra lire une variable seule, de type quelconque, ou un ensemble de variables séparées par des virgules, de type identique ou différent.

- L'édition de résultats sur un périphérique de sortie

Algorithme	Algorigramme (NF Z 67-010)
Ecrire();	

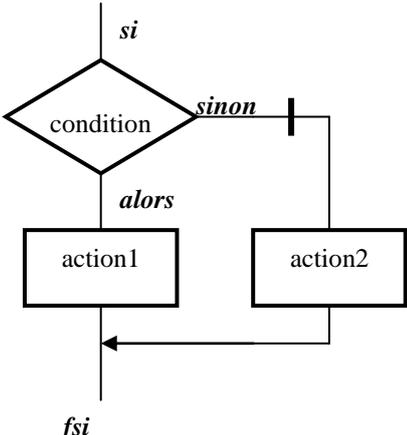


Remarque.: On pourra écrire les valeurs d'une ou plusieurs variables.

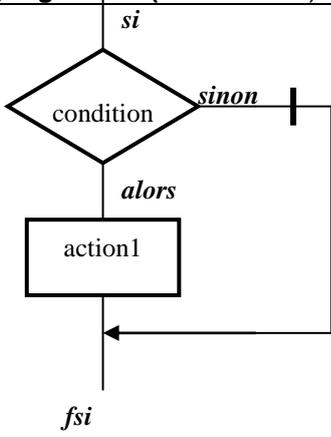
3.2 Les structures alternatives

- La structure alternative de base

Selon qu'une condition est vraie ou fausse, **deux traitements sont possibles.**

Algorithme	Algorigramme (NF Z 67-010)
<pre> si (condition vraie) alors action1 ; sinon action2 ; fsi </pre>	

Si **action2** est vide, on écrira :

Algorithme	Algorigramme (NF Z 67-010)
<pre> si (condition vraie) alors action1 ; sinon rien ; fsi </pre>	

3.3 Les structures itératives (ou répétitives)

Ces structures permettent d'exécuter plusieurs fois une séquence d'instructions.

3.3.1 Le nombre d'itérations (répétitions) est connu

- **Boucle de comptage**

Lorsque le nombre d'itérations est connu, on peut utiliser une variable auxiliaire (**compteur de boucle**) dont la valeur caractérise le nombre de passages dans la boucle.

Algorithme	Algorithme (NF Z 67-010)
<pre> // Compteur de boucle i : entier; pour i variant de <valeur_initiale> jusqu'à <valeur_finale> par pas de <n> faire Action(s) fpour; </pre>	<pre> graph TD Start(()) --> Init[i ← <valeur_initiale>] Init --> Action[Action(s)] Action --> Inc[i ← i + n] Inc --> Dec{i ≥ <valeur_finale>} Dec -- faux --> Loop(()) Loop --> Init Dec -- vraie --> End(()) </pre>

EXERCICES D'ALGORITHMIQUE

Des exemples et des exercices dans « **Le cahier d'exercices du cours d'algorithmique** »

3.3.2 Le nombre d'itérations (répétitions) est inconnu

- Test en tête de boucle

Algorithme	Algorithme (NF Z 67-010)
<p>tant que (<i>condition</i>) faire <i>Action(s)</i> tant que;</p>	

- Test en fin de boucle

Algorithme	Algorithme (NF Z 67-010)
<p>faire <i>Action(s)</i> tant que (<i>condition</i>) ;</p>	

TRES IMPORTANT

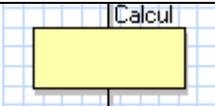
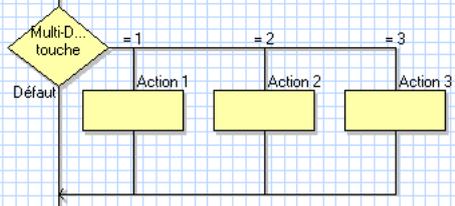
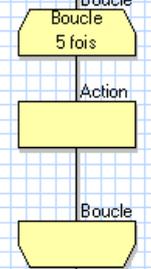
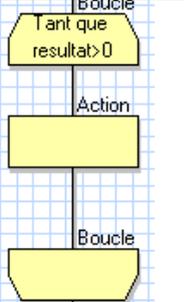
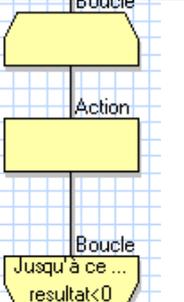
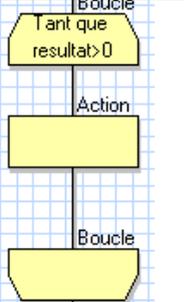
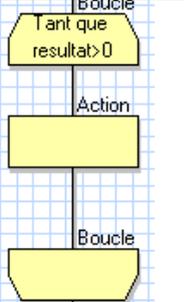
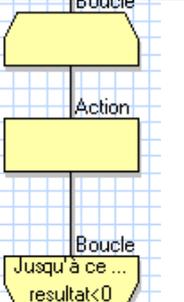
Pour que la boucle puisse finir, il faut que l'exécution de l'action modifie la valeur de l'expression conditionnelle évaluée pour le test de sortie de la boucle.

Il faut veiller à **initialiser les variables** intervenant dans la condition avant l'arrivée à la boucle, sinon son évaluation pourrait conduire à une sortie aléatoire de la boucle.

**EXERCICES
D'ALGORITHMIQUE**

Des exemples et des exercices dans « **Le cahier d'exercices du cours d'algorithmique** »

Annexe : Symboles particuliers dans « Flowcode » et « Visual Programming Language »

Algorithmique	Logiciel Flowcode	Langage VPL			
Commentaires					
Constante	Idem norme NFZ 67-010				
Variable					
Affectation		Pas de symbole particulier			
Choix	Idem norme NFZ 67-010				
Choix multiple					
Boucle « pour... (Nombre d'itération connu)		Pas de symbole particulier			
Boucles (Nombre d'itération inconnu)	<table border="1" style="width: 100%;"> <tr> <th>Tant que ... faire</th> <th>Faire ... Tant que</th> </tr> <tr> <td></td> <td></td> </tr> </table>		Tant que ... faire	Faire ... Tant que	
	Tant que ... faire	Faire ... Tant que			
					

Index

1	Présentation : démarche adoptée pour le développement d'un programme	1
	Phase 1 : CDCF	
	Phase 2 : Algorithmique	2
	Phase 3 : Codage	
	Exemple	3
2	Organisation d'un algorithme	
	2.1 L'en-tête	5
	2.2 La partie déclarative	
	2.3 La partie exécutive	
	2.4 Les commentaires	6
	2.5 Le renvoi	
3	<u>Les « briques » d'un algorithme</u>	
	3.1 Les actions de base	6
	L'affectation	
	La saisie d'une valeur sur un terminal	
	L'édition de résultats sur un périphérique	
	3.2 Les structures alternatives	7
	La structure alternative de base	
	Les structures alternatives imbriquées	8
	La structure de choix multiple	
	3.3 Les structures itératives ou répétitives	9
	Le nombre d'itérations est connu	
	Le nombre d'itérations est inconnu	10
	Test en fin de boucle	
	Test en début de boucle	
	Annexe : Symboles « Flowcode » et « Visual Programming Language » particuliers	11