**1****) Présentation**



**Calculateur**

**Exécution**

**Programme machine**

**CdCF**

Ecriture du Programme en Langage évolué

**Compilation-Assemblage**

**Programme exécutable**

**ALGORITHME** **ou**

**ALGORIGRAMME**

**Transcription**

**Analyse du problème**

**Programmation**

**Tests**

**Phase 1**

**Phase 2**

**Phase 3**

Le développement d’un programme nécessite **trois phases.**

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

**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.

**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**’algorigramme**)

**PRINCIPAUX SYMBOLES UTILISES**

(NF Z 67-010)



**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 !)

## 2 Développement du programme et mise au point.

Que ce soit dans un langage assembleur ou évolué, l’écriture du programme ainsi que sa mise au point doivent suivre le diagramme suivant :



L’émulateur est un dispositif (assez onéreux) qui permet de remplacer le microcontrôleur afin d’effectuer la vérification et la mise au point du programme. C’est une carte électronique comportant, d’un coté un connecteur compatible avec le boîtier du microcontrôleur qu’il émule, et de l’autre, une connexion de type RS232 (ou autre) reliée à micro-ordinateur.

La mise au point peut alors se faire en pas à pas ou avec des points d’arrêt permettant ainsi de cerner très rapidement les “bugs” de certaines routines.

Le simulateur , beaucoup moins coûteux, permet la même chose mais de façon beaucoup moins efficace car il ne tient pas ou très peu compte de l’environnement du microcontrôleur

**2** **Organisation d’un algorithme**



**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*

**Pi**

***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) |
| **début**  ....  ...  **fin.** | **début**  **fin** |

**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**

1. **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 **** .



Exemples : x  5 x  x + 1

|  |  |
| --- | --- |
| **Algorithme** | **Algorigramme** (NF Z 67-010) |
| *Nom\_Variable* *****valeur* |  |

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

1. **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.

1. **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**

1. **La structure alternative de base**

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

|  |  |
| --- | --- |
| **Algorithme** | **Algorigramme** (NF Z 67-010) |
| **si** (*condition vraie)* **alors** *action1 ;*  **sinon** *action2 ;*  **fsi** | condition  action1  ***sinon***  ***alors***  ***si***  action2  ***fsi*** |

Si **action2** est vide, on écrira :

|  |  |
| --- | --- |
| **Algorithme** | **Algorigramme (**NF Z 67-010)  condition  action1  ***sinon***  ***alors***  ***si***  ***fsi*** |
| **si** (*condition vraie)* **alors** *action1 ;*  **sinon** *rien ;*  **fsi** |  |



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

1. **Les structures alternatives imbriquées**

Il peut arriver que « action1 » et « action2 » soient elles-mêmes des actions comportant un choix, par exemple :

**si** (*condition1*) **alors** **si** (condition2) **alors** *action1.1 ;*

**sinon** *action1.2 ;*

**fsi**

**sinon** **si** (condition3) **alors** *action2.1 ;*

**sinon** *action2.2 ;*

**fsi**

**fsi**

Remarque : Si les alternatives sont complètes, il n’y a aucune difficulté de compréhension; par contre, si certaines sont vides, il faut respecter la règle suivante : **chaque « sinon » se rapporte au dernier « si » non encore complété que l’on rencontre**.

1. **Structure de choix multiple**

Lorsqu’on peut choisir entre plusieurs possibilités se rapportant à une **même** expression, on écrit :

|  |  |
| --- | --- |
| **Algorithme** | **Algorigramme** (NF Z 67-010) |
| **selon** (*expression)*  *valeur 1 : action\_1;* ***fselon;***  *valeur 2 : action\_2;* ***fselon;***  *…*  *valeur n : action\_n*;***fselon;***  **autrement** : *action\_0 ;*  **fselon**; | ***autrement***  ***fselon***  valeur 1  action 1  Action 0  **selon** (*expression)*  ***alors***  ***sinon***  valeur 2  action 2  valeur n  action n |

**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**

1. **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** | **Algorigramme** (NF Z 67-010) |
| // Compteur de boucle  i : entier;    **pour i** variant de *<valeur\_initiale>* jusqu’à *<valeur\_finale>* par pas de <*n*>  **faire**  *Action(s)*  **fpour;** | i  <valeur\_initiale>  Action(s)  **i  i + n**  i  <valeur\_finale>  *vraie*  *faux* |



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

**3.3.2** **Le nombre d’itérations (répétitions) est inconnu**

1. **Test en tête de boucle**

|  |  |
| --- | --- |
| **Algorithme** | **Algorigramme** (NF Z 67-010) |
| **tant que** (*condition*)  **faire**  *Action(s)*  **ftantque**; |  |

1. **Test en fin de boucle**

|  |  |
| --- | --- |
| **Algorithme** | **Algorigramme** (NF Z 67-010) |
| **faire**  *Action(s)*  **tant que** (*condition) ;* |  |

**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**.**