C — Ch 1 – Introduction

I. Introduction aux langages

Rappel : l'activité de programmation (ou codage) consiste à exprimer la résolution d'un problème à l'aide des instructions de base d'un langage de programmation. **Avant de programmer, les étapes de résolution du problème ont donc dû être matérialisées en français ou dans le langage simplifié de l'algorithmique**.

A. Notion de programme

Un **PROGRAMME SOURCE** est le résultat de traduction d'un algorithme (exprimé en pseudo-code, ou sous forme graphique) dans un langage de programmation. Il est stocké dans un fichier au format texte dont l'extension dépend du langage utilisé (en C : .c, en C++ : généralement .cpp).

Synonyme : code source

Un **PROGRAMME COMPILE** est le résultat de la traduction d'un programme source (exprimé dans un langage de programmation) en un langage exécutable par l'ordinateur, le langage machine. Il est stocké dans un fichier binaire, son extension est .exe (sous Windows).

Synonyme : exécutable

Une **APPLICATION INFORMATIQUE** est un programme ou un ensemble de programmes utilisables par l'utilisateur pour l'aider dans sa tâche.

Synonymes : logiciel

B. Langages de programmation

Les langages de programmation ont été conçus afin de répondre :

- A **l'évolution des technologies** liées à l'informatique (puissance des ordinateurs, capacité de la mémoire vive) :
  - au départ, les langages ont été très liés à l'aspect matériel des ordinateurs (binaire, le courant passe ou ne passe pas),
  - puis, l'expression s'est voulue plus proche du langage naturel ;
- A **des besoins particuliers** : gestion, calcul scientifique, enseignement, systèmes d'exploitation, graphisme, intelligence artificielle, etc.
- A **l'évolution dans la manière d'appréhender les problèmes** et de concevoir les programmes (on parle de « **paradigmes** de programmation »).

C. Familles de langages de programmation

On peut classer les langages de programmation selon plusieurs critères pour constituer des familles de langages.

Voici quelques unes des caractéristiques fréquemment utilisées pour classer un langage :

- sa proximité ou son éloignement du langage machine : générations
- la manière de concevoir les programmes dans ce langage ;
- le mode d'exécution des programmes ;

Ces classements ne sont pas strictement définis mais permettent de catégoriser les langages..

1. Générations de langages

Génération	**Caractéristique**	**exemples**
L1G : Langages de 1ère génération	Correspond au langage machine : chaque instruction machine est codée sous forme binaire, une succession de 0 et de 1	C'est le langage de base des microprocesseurs (jeu d'instructions) **Exemple :** 000000 0000 101 001
L2G : Langages de 2nde génération	à chaque instruction machine, on associe un code mnémonique, plus simple à retenir qu'une suite de 0 et de 1	Assembleurs **Exemple :** ADD R1, R5
L3G : Langage de 3ème génération	La syntaxe des L3G se rapproche du langage naturel (exprimé sous forme de pseudo-code) On parle de langage de haut niveau	Cobol, C, Pascal, C++, Java, C#, Visual Basic, Ada, etc. **Exemple :** compute a = (b+c).
L4G : Langages de 4ème génération	La syntaxe est de plus haut niveau encore : quelques instructions suffisent à construire des programmes complets de manière très rapide On parle de RAD pour Rapid Application Development.	W-langage, le langage de Windev, Application Master, autrefois L4G des mainframes ICL) **Exemple :** MatRéelAdditionne(NomMatrice, 5)

2. Paradigmes de programmation[1]

Manière concevoir la programmation	**caractéristique**	**Exemples de langages**

[1] Paradigmes de programmation : manière « de voir les choses » en programmation

PROGRAMMATION IMPERATIVE	Des instructions manipulent des données constituant l'état du programme et font évoluer ces valeurs vers le résultat attendu (description du **comment** faire)	.
Programmation Procédurale ou Structurée	*On conçoit un programme comme un ensemble de procédures et fonctions qui vont modifier les variables d'état du programme.*	*C, Cobol, Pascal, Basic Fortran* ***PHP*** ***C++**, Cobol, Pascal, Java, C#, Visual Basic* *Bash*
Programmation à Objets	*On conçoit un programme comme un ensemble d'objets échangeant des messages.*	*Eiffel, Smalltalk, **Java*** ***C++**, Cobol Objet, Pascal Objet (Delphi), C#, VB.net, **PHP version 5***
Programmation Fonctionnelle	*Des fonctions manipulent des ensembles de données constitués par d'autres fonctions (pas d'effet de bord)*	*Scheme, Lisp*
PROGRAMMATION DECLARATIVE	Des déclarations décrivent le problème (description de **quoi** faire)	
Programmation Déclarative	*Des déclarations décrivent un ensemble de règles à appliquer*	*XSLT*
Programmation Logique	*Un programme est une suite de déclaration de faits et de règles ; un « moteur d'inférence » déduit des conclusions (nouveaux faits) à partir de ces règles*	*Prolog*

3. Modes d'exécution des programmes

Mode exéc.	Caractéristique	exemples
interprétés	Les instructions du programme source sont exécutées par un programme interpréteur : celui-ci lit chaque instruction et la traduit en instruction machine (au fur et à mesure) **L'exécution est lente**	**PHP, Javascript**, VBscript, Bash Logo
compilés	Le programme source est compilé **L'exécution est très rapide**	C, **C++**, Pascal
semi-compilés	Le source est traduit en un code semi-compilé (on parle de ***bytecode*** - non directement exécutable par la machine). Un programme particulier, appelé « machine virtuelle » (*en anglais : **VM**, Virtual Machine*) se charge de convertir le bytecode en langage machine au moment de l'exécution. Un « ramasse-miettes » (*en anglais garbage collector*) nettoie la mémoire **L'exécution est rapide**	**Java** C#, VB.net, etc.

.

D. Programmes et types d'interactions

Un programme peut être classé en termes de modes d'interactions qu'il utilise (ce critère n'est pas lié à un langage).

Interactions avec l'utilisateur :

Programmes	**caractéristique**	**exemples**
En mode Console, en traitement par lots (Batch)	Un programme exécute un traitement avec très peu d'interactions avec l'utilisateur ou sans interaction pour le batch. A chaque moment, on connaît la prochaine instruction qui sera exécutée	Apprentissage, Calculs longs et complexes
Evènementielles	L'exécution du programme est pilotée par les évènements déclenchés par les interactions de l'utilisateur avec l'interface graphique. La prochaine instruction dépendra de l'action de l'utilisateur (click sur tel ou tel bouton, etc., = non prévisible)	Suites bureautiques, Logiciels de gestion, de dessin, etc. = les applications actuelles

Interactions entre programmes :

Client-Serveur	applications faisant appel à des services distants	navigateur et serveur Web, client FTP et serveur FTP, application et serveur de données (SGBD)

Les applications en mode console et évènementielles peuvent également être du type client–serveur, c'est-à-dire faire appel à des services distants.

E. Critères qualitatifs des logiciels

Un certain nombre de règles (*anglais : rules*) d'écriture des programmes vont permettre de construire des applications de qualité.

Quelques uns des aspects qualitatifs d'une application :

- La **maintenabilité** (*anglais maintainability*) : qualité d'un programme source à être modifié ; qualité du code source à être lisible – lisibilité (*anglais : readability*) (=utiliser des commentaires, des noms de variables clairs, créer des modules indépendants, etc.)
- La **fiabilité** (*anglais : reliability*) : comportement prévisible (= répondre exactement aux besoins de l'utilisateur, prévoir les erreurs et les gérer, etc.) ;
- La **performance** (*anglais performance*) (=optimiser les calculs, etc.)
- La **sécurité** (*anglais : security*) : en conflit avec la performance dans la mesure où elle va nécessiter l'ajout de code source supplémentaire, cette qualité est néanmoins fondamentale (=ajouter du code de gestion des erreurs de saisie, etc.);
- La **portabilité** (*anglais : portability*) (=écrire sans utiliser des particularités spécifiques à l'ordinateur utilisé, etc.)
- la **disponibilité** (*anglais : availability*)

L'écriture du code source devra tenir compte de ces règles afin de construire des applications de qualité.

Cf.

https://www.securecoding.cert.org/confluence/display/seccode/CERT+Secure+Coding+Standards

F. Etapes de programmation et outils associés

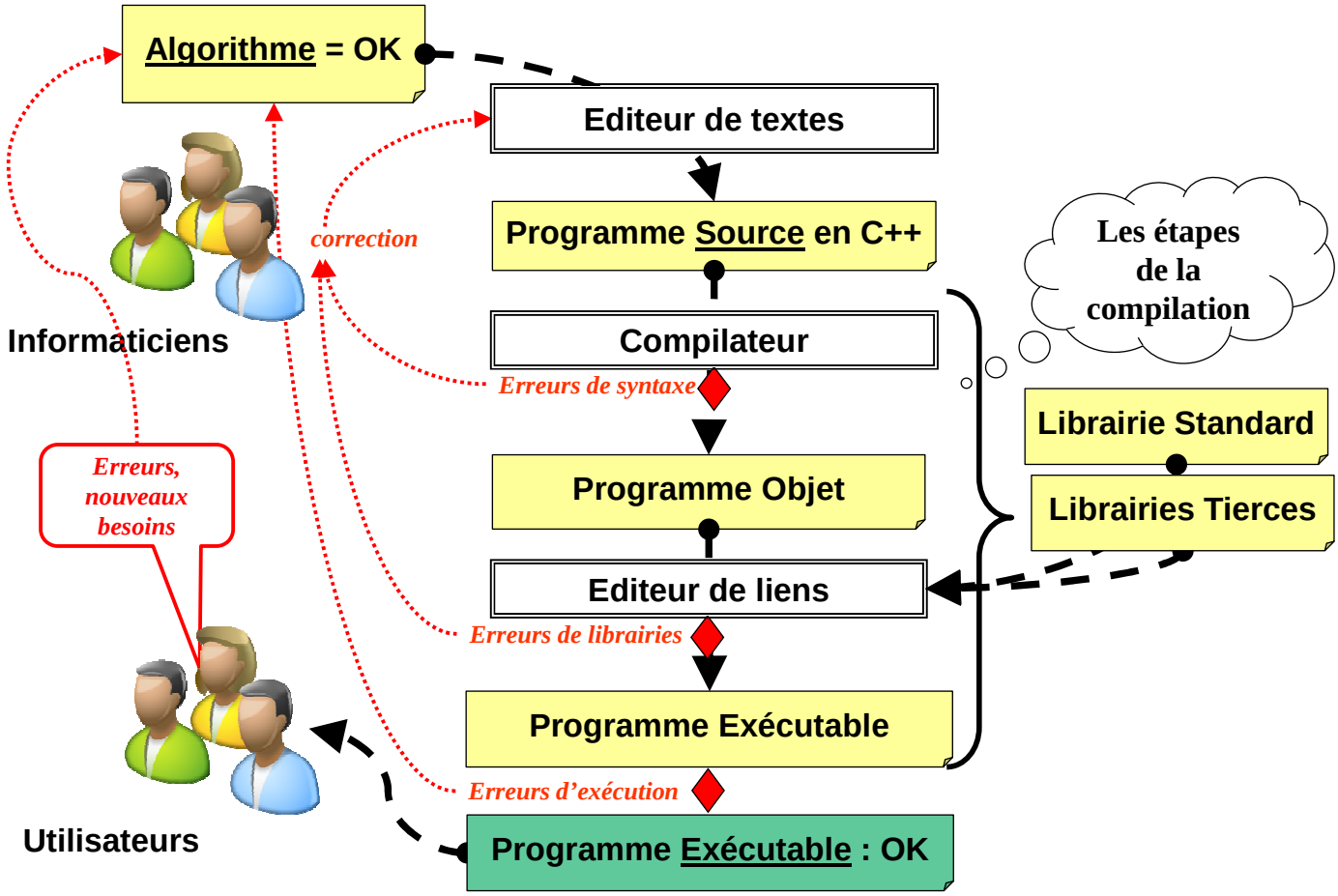

Figure 1 : les étapes de construction d'une programme informatique

1. Editeur de texte

Un éditeur de texte est un logiciel permettant d'écrire un fichier texte (suite de caractères sans mise en forme, non formaté).

Un bon éditeur de texte, pour la programmation, doit offrir :

- Obligatoirement la **coloration syntaxique** : coloration des mots-clefs du langage de programmation ;
- éventuellement la **complétion de code** : proposition de syntaxe des instructions en fonction du contexte de saisie ;

Exemples :

- Pspad sous Windows, Gedit sous Linux, Scite, Notepad++
- emacs, Vim (Linux) et Gvim (Windows)

2. Compilateur du langage vers le langage machine

Le compilateur traduit le langage source, exprimé dans un langage de programmation, en langage machine.

En fait, la « compilation » est un processus plus complexe qu'une simple traduction, et se déroule en 3 phases essentielles, réalisées par les programmes suivants :

- Le **pré-processeur** (pre-processor) : analyse des directives de compilation (ex : #include pour les langages C et C++), suppression des commentaires.

- Le **compilateur** (compiler) : il assure la traduction du code source en code machine en tenant compte du jeu d'instruction de la famille de processeur, du mode d'adressage (analyse la syntaxe du programme source et produit un programme compilé, mais non exécutable)
- L'**éditeur de liens** (linker) : met en relation le programme avec les librairies externes utilisées et construit le programme exécutable.

3. EDI, anglais IDE

L'**EDI** (Environnement de Développement Intégré, en anglais : **IDE**, Integrated Development Environment) est un outil intégrant toutes les fonctions permettant l'écriture des sources, la compilation et le débogage[2].

Il inclut également des outils permettant la construction d'interfaces graphiques et de nombreuses bibliothèques de procédures et fonctions permettant de produire rapidement des applications.

Exemples :

- Pour C/C++ : Dev C++, Anjuta, Code::Blocks
- Pour Ada : Adagide
- Pour Java : NetBeans
- Pour Pascal Objet : Borland Delphi
- Pour VB.net et C# : Microsoft Visual Studio
- Paramétrable (plugins) : Eclipse (Open Source), IBM WSAD (commercial)

4. AGL

L'**AGL** (Atelier de Génie Logiciel) intègre toutes les phases de développement des applications à l'échelle de l'entreprise :

- Analyse et modélisation, conception des bases de données
- Un environnement de développement intégré permettant le travail en équipes de programmeur
- Des outils de tests et de distribution de logiciels.

Il utilise souvent un langage propriétaire (spécifique).

Exemple :

- Windev

II. Le langage C

A. Eléments généraux

Le langage C a été conçu par Dennis Ritchie, chercheur aux laboratoires Bell dans les années 1970 afin d'écrire le système d'exploitation Unix. Le langage se devait d'être à la fois proche de la machine et de haut niveau, et devait être simple d'utilisation.

Il est normalisé au début des années 1990 : il devient ISO C 90 (appelé aussi ANSI C90), puis des améliorations sont incluses dans la version ISO C 99 (afin d'assurer, autant que possible, la portabilité de C vers C++) . La version actuelle du standard (2012-2013) est la version C11 (C1X). Les implémentations des compilateurs tendent à intégrer les améliorations apportées (document de travail en ligne C99 http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1124.pdf , et C11 http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1570.pdf).

[2] Debogage : recherche et correction des défauts de conception du programme conduisant à des disfonctionnements ou bogues (en anglais : bugs)

Cf. options du compilateur GCC pour définir le niveau de prise en compte des différents standards : http://gcc.gnu.org/onlinedocs/gcc/Standards.html

Le C est un langage de haut niveau, typé (il intègre les types de données de base), des structures de contrôles et des structures de données (tableaux, enregistrements). Des bibliothèques standard fournissent de nombreuses fonctions : entrées, sorties, gestion de la mémoire, etc.

La portabilité du langage C (« pouvoir compiler un code source aussi bien sous Linux que sous Windows », par exemple), est cependant toute relative, de nombreux compilateurs incorporant en effet des bibliothèques spécifiques. Il faut donc s'attacher à n'utiliser que les éléments définis dans les standards ou des bibliothèques tierces portables.

Ouvrage de référence : « The C programming Language », 1978, Brian Kernighan et Dennis Ritchie – K & R (attention : la norme a évoluée depuis).

B. Exemple « helloworld »

```
(1)  /* hello.c */
(2)  #include <stdio.h>
(3)  int main()
(4)  {
(5)       printf("hello World !") ; // afficher hello World
(6)       return 0 ;
(7)  }
```

(1) Un commentaire 'bloc'

(2) inclusion d'un fichier contenant la déclaration de fonctions pour gérer les entrées/sorties standard

(3) début de la fonction 'main' (point d'entrée de tout programme), de type entier

(4) ouverture du bloc de la fonction main

(5) affichage du texte 'hello World !' , suivi d'un commentaire 'ligne'

(6) quitte la fonction main en renvoyant la valeur 0 (en général, signifie que 'tout s'est bien passé')

(7) fermeture du bloc de la fonction main