

Christoph Samuel

La programmation informatique

Avril 2021

Sommaire

1	Introduction	2
2	Une brève histoire de la programmation	2
2.1	La fin des programmeurs?	3
3	Pratique	4
4	Phases de création d'un programme	4
4.1	Conception	5
4.2	Codage	5
4.3	Transformation du code source	5
4.3.1	Compilation	5
4.3.2	Interprétation	6
4.3.3	Avantages, inconvénients	7
4.4	Test du programme	7
5	Techniques de programmation	8

Table des figures

1	Programmation informatique	2
2	Machine métier a tisser	2
3	Métier a tisser version dessin	3
4	Le codage	5
5	Exemple de compilation	6
6	Un exemple de programmation en Python	6
7	Schéma bilan sur la transformation du code source	7

Liste des tableaux

1	Classement des meilleurs langages de programmation depuis 2005	7
---	--	---

Résumé

Le document porte sur la programmation informatique, dans un premier temps nous verrons une brève histoire sur la programmation² ensuite nous verrons ses différentes pratiques³, pour enfin finir sur la création un programme informatique et les différents types de programmation⁵ dans l'informatique.

1 Introduction

La programmation dans le domaine informatique est l'ensemble des activités qui permettent l'écriture des programmes informatiques. C'est une étape importante de la conception de logiciel (voire de matériel). Pour écrire le résultat de cette activité, on utilise un *langage de programmation*¹.

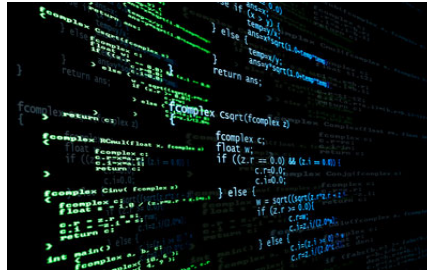


FIGURE 1 – Programmation informatique

La programmation représente le plus souvent le codage, c'est-à-dire la rédaction du code source d'un logiciel. On utilise plutôt le terme développement pour lister l'ensemble des activités liées à la création d'un logiciel.

2 Une brève histoire de la programmation

La première machine programmable (c'est-à-dire machine dont les possibilités changent quand on modifie son "programme") est probablement le métier à tisser de Jacquard, qui a été réalisé en 1801. La machine utilisait une suite de cartons perforés. Les trous indiquaient le motif que le métier suivait pour réaliser un tissage.

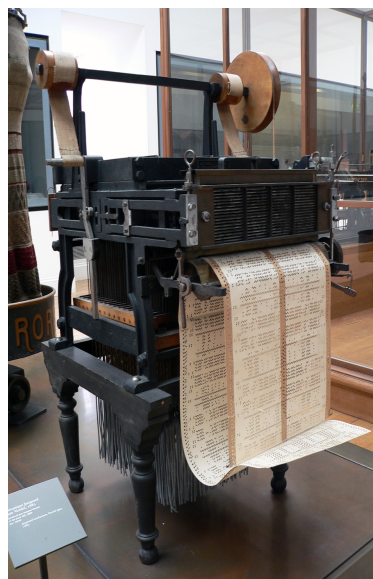


FIGURE 2 – Machine métier à tisser

1. notation conventionnelle destinée à formuler des algorithmes et produire des programmes informatiques qui les appliquent

Cette innovation a été ensuite améliorée par Herman Hollerith d'IBM pour le développement de la fameuse carte perforée d'IBM.



FIGURE 3 – Métier à tisser version dessin

En 1936, la publication de *On Computable Numbers with an Application to the Entscheidungsproblem*² par Alan Mathison Turing donne le coup d'envoi à la création de l'ordinateur programmable. Il y présente sa machine de Turing, le premier calculateur universel programmable, et invente les concepts et les termes de programmation et de programme. Les programmes devenant plus complexes, cela est devenu presque impossible, parce qu'une seule erreur rendait le programme entier inutilisable. Avec les progrès des supports de données, il devient possible de charger le programme à partir de cartes perforées, contenant la liste des instructions en code binaire spécifique à un type d'ordinateur particulier. La puissance des ordinateurs augmentant, on les utilisa pour faire les programmes, les programmeurs préférant naturellement rédiger du texte plutôt que des suites de 0 et de 1, à charge pour l'ordinateur d'en faire la traduction lui-même. Avec le temps, de nouveaux langages de programmation sont apparus, faisant de plus en plus abstraction du matériel sur lequel devaient tourner les programmes. Ceci plusieurs facteurs de gains : ces langages sont plus faciles à apprendre, un programmeur peut produire du code plus rapidement, et les programmes produits peuvent tourner sur différents types de machines.

2.1 La fin des programmeurs ?

De tous temps, on a prédit « la fin des programmeurs ». Dans les années 60, les langages symboliques comme AUTO-CODE, Cobol et Fortran ont en effet mis fin "en grande partie" à la programmation de bas niveau tel que l'assembleur³. Il semblait alors clair que n'importe qui était capable d'écrire du code du type :

```
multiply MONTANT-HT by TAUX-TVA giving MONTANT-TAXES.      add MONTANT-HT,  
MONTANT-TAXES giving MONTANT-TTC.
```

ou

2. article fondateur de la science informatique

3. langage de plus bas niveau qui représente le langage machine sous une forme lisible par un humain.

$$RDELTA = \sqrt{B^2 - 4AC} \quad X1 = (-B + RDELTA) / (2A)$$

plutôt que des dizaines de lignes comme

```

    movl    %esp, %ebp    subl    $24, %esp    flds    12(%ebp)    fmul    %st, %st(1)
12(%ebp)    flds    8(%ebp)    flds    .LC0    fmul    %st, %st(1)    fmul    %st, %st(1)
16(%ebp)    fsub    %st, %st(1)    fstpl    (%esp)    call    sqrt    fsts    %st(0), %st
-4(%ebp)    flds    -4(%ebp)    fsub    12(%ebp)    flds    8(%ebp)    fadd
    fdiv    %st, %st(1)

```

Pourtant il a vite fallu se rendre compte que la programmation ne se limitait pas au codage, et que la conception d'applications était un vrai métier qui ne s'improvise pas.

3 Pratique

En programmation informatique la pratique est établie sous plusieurs points parmi lesquels on retrouve l'algorithmique, la gestion de versions, l'optimisation du code, la programmation système, le refactoring et des tests (intégration et unitaire).

1. L'algorithmique est l'étude et la production de règles et techniques impliquées dans la définition et la conception d'algorithmes.
2. La gestion de versions consiste à gérer l'ensemble des versions d'un ou plusieurs fichiers, elle concerne surtout la gestion des codes source.
3. L'optimisation de code consiste à améliorer l'efficacité du code informatique d'un programme (plus rapide, moins de place mémoire...)
4. La programmation système est un type de programmation qui vise au développement de programmes qui font partie du système d'exploitation d'un ordinateur ou qui en réalisent les fonctions.
5. Le refactoring⁴ est l'opération consistant à retravailler le code source d'un programme informatique sans y ajouter des fonctionnalités de façon à en améliorer la lisibilité.
6. Le test d'intégration est une phase de tests, vérifiant le bon fonctionnement d'une partie précise d'un logiciel.

4 Phases de création d'un programme

On retrouve dans la programmation informatique une étape cruciale qui est la création d'un programme. Cette création s'effectue en 4 phases, une phase de conception suivie d'une phase de codage puis une phase de transformation du code source et pour finir le test du programme final.

4. réusinage de code

4.1 Conception

La phase de conception définit le but du programme. Si on fait une rapide analyse fonctionnelle d'un programme, on détermine essentiellement les données qu'il va traiter (données d'entrée), la méthode employée (l'algorithme), et le résultat (données de sortie). Les données d'entrée et de sortie peuvent être de nature très diverses. On retrouve en général les mêmes fonctionnalités de base :

Pour la programmation impérative

1. Si alors (SI prédicat ALORS faire ceci SINON faire cela),
2. Tant que (TANT QUE prédicat faire...),
3. Pour (Pour variable allant de borne inférieur à borne supérieur faire ...),

4.2 Codage

Une fois l'algorithme défini, l'étape suivante est de coder le programme. Le codage dépend de l'architecture sur laquelle va s'exécuter le programme, de compromis temps-mémoire, et d'autres contraintes. Ces contraintes vont déterminer quel langage de programmation utiliser pour "convertir" l'algorithme en code source.

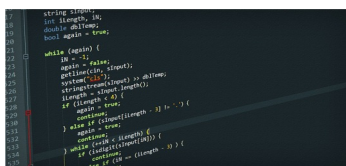


FIGURE 4 – Le codage

4.3 Transformation du code source

Le code source⁵ n'est (presque) jamais utilisable tel quel. Il est généralement écrit dans un langage "de haut niveau", compréhensible pour l'homme, mais pas pour la machine.[2]

4.3.1 Compilation

Certains langages sont ce qu'on appelle des langages compilés[4]. En toute généralité, la compilation est l'opération qui consiste à transformer un langage source en un langage cible. Dans le cas d'un programme, le compilateur va transformer tout le texte représentant le code source du programme, en code compréhensible pour la machine, appelé code machine. Dans le cas de langages dits compilés, ce qui est exécuté est le résultat de la compilation. Une fois effectuée, l'exécutable obtenu peut être utilisé sans le code source.

5. texte qui présente les instructions composant un programme sous une forme lisible

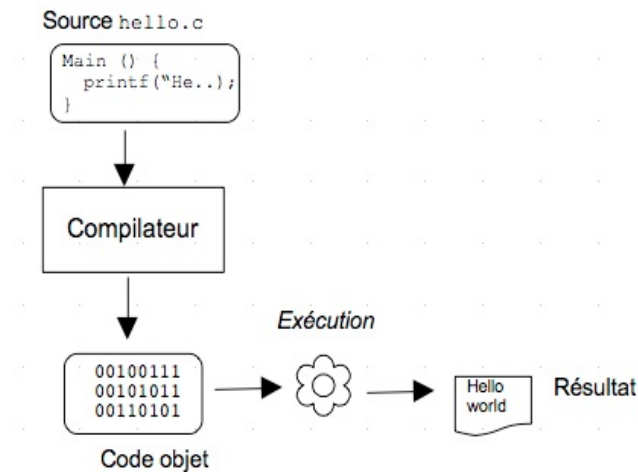


FIGURE 5 – Exemple de compilation

Il faut également noter que le résultat de la compilation n'est pas forcément du code machine correspondant à la machine réelle, mais peut être du code compris par une machine virtuelle (c'est-à-dire un programme simulant une machine), auquel cas on parlera de bytecode⁶. C'est par exemple le cas en Java⁷. L'avantage est que, de cette façon, un programme peut fonctionner sur n'importe quelle machine réelle, du moment que la machine virtuelle existe pour celle-ci.

4.3.2 Interprétation

D'autres langages ne nécessitent pas de phase spéciale de compilation. La méthode employée pour exécuter le programme est alors différente. Le programme entier n'est jamais compilé. Chaque ligne de code est compilée « en temps réel » par un programme. On dit de ce programme qu'il interprète le code source. Par exemple, python ou perl sont des langages interprétés.

```

1 from os import getpid
2
3
4 class processus:
5     """ Nouvelle classe Python """
6     def __init__(self):
7         self.pid = 1
8
9     def get_next_process_id(self):
10        try:
11            p1 = getpid()
12            p2 = p1 + 1
13            print("Le processus suivant est: %d" % p2)
14            return p2
15        except Exception as e:
16            print("Exception: " + str(e))
17            return self.pid
  
```

FIGURE 6 – Un exemple de programmation en Python

6. code intermédiaire entre les instructions machines et le code source, qui n'est pas directement exécutable

7. Java est un langage de programmation orienté objet

langage de programmation	rang 2020	rang 2015	rang 2010	rang 2005
Java	1	2	1	2
C	1	1	2	1
Python	3	7	6	6
C++	4	4	4	3

TABLE 1 – Classement des meilleurs langages de programmation depuis 2005

Cependant, ce serait faux de dire que la compilation n'intervient pas. L'interprète produit le code machine, au fur et à mesure de l'exécution du programme, en compilant chaque ligne du code source.

4.3.3 Avantages, inconvénients

Ces langages comportent tous des avantages, des inconvénients et leur domaines de pratique on retrouve donc des études et ouvrages pour chacun d'eux par exemple avec The C programming language[1] pour le C. Les avantages généralement retenus pour l'utilisation de langages "compilés", est qu'ils sont plus rapides à l'exécution que des langages interprétés, car l'interprète doit être lancé à chaque exécution du programme, ce qui mobilise systématiquement les ressources.

Traditionnellement, les langages interprétés offrent en revanche une certaine portabilité (la capacité à utiliser le code source sur différentes plates-formes), ainsi qu'une facilité pour l'écriture du code. En effet, il n'est pas nécessaire de passer par la phase de compilation pour tester le code source. Les quatre langages1 vu précédemment sont les plus utilisés.

4.4 Test du programme

C'est l'une des étapes les plus importantes de la création d'un programme. En principe, tout programmeur se doit de vérifier chaque partie d'un programme, de le tester. Il existe différents types de test.

On peut citer en particulier :

1. Test unitaire
2. Test d'intégration
3. Test de performance

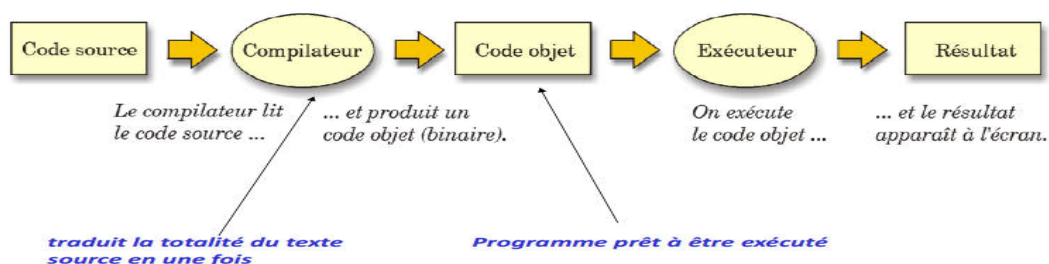


FIGURE 7 – Schéma bilan sur la transformation du code source

Il convient de noter qu'il est parfois possible de vérifier un programme informatique, c'est-à-dire prouver, de manière plus ou moins automatique, qu'il assure certaines propriétés.

5 Techniques de programmation

On retrouve de nombreuses techniques de programmation par exemple la programmation par objet qui consiste à utiliser des techniques de programmation pour mettre en œuvre une conception basée sur les objets. Celle-ci peut être élaborée en utilisant des méthodologies de développement logiciel objet, la programmation concurrente qui est un paradigme de programmation tenant compte, dans un programme, de l'existence de plusieurs piles sémantiques qui peuvent être appelées threads, processus ou tâches. Elles sont matérialisées en machine par une pile d'exécution et un ensemble de données privées, ou encore comme la programmation impérative vu précédemment.

Voici une liste de quelques autres techniques de programmation que l'on retrouve dans la programmation informatique

- Programmation fonctionnelle
- Programmation logique
- Programmation orientée prototype
- Programmation par contraintes [5]
- Programmation structurée

Références

- [1] Brian Hernigan. Denmis Ritchie. *The C programming language*. Prentice Hall, 1978.
- [2] J.FRANCIS. J.Fuegi. Lovelace et bablage and the creationof the 1843 "notes". *DOI*, 25, 2003.
- [3] Alan Mathison Turing. *On computable numbers with an application to the Entscheidungsproblem*. London Mathematical Society, 1936.
- [4] Utilisateur. Histoire des langages de programmation. *Wikipédia*, 2021.
- [5] Vincent Vigneron. *Programmation par contraintes et découverte de motifs sur données séquentielles*. PhD thesis, Standfort University, 2017.