https://laurentbloch.net/BlogLB/Pour-l-amour-d-Ada

Pour l'amour d'Ada

Les chats ne sont pas de gauche -

Date de mise en ligne : mardi 25 octobre 2022

Chapitre précédent[/Chapitre suivant/]

Au chapitre L'informatique, tentative d'explication j'ai écrit que pour obtenir d'un ordinateur qu'il fournisse un résultat il fallait écrire un programme, et qu'il existait pour ce faire des langages de programmation. On peut choisir un langage de programmation parce qu'il est bien adapté au problème à résoudre, mais les questions de tournure d'esprit du programmeur, voire ses goûts personnels, jouent aussi leur rôle. Les controverses relatives aux langages sont l'une des principales distractions des informaticiens.

Quel que soit le langage, le programme, pour être exécuté, doit *in fine* être traduit dans le langage machine de l'ordinateur utilisé, le seul que cet ordinateur puisse interpréter. Le programme qui effectue cette traduction s'appelle un compilateur. Pour chaque langage de programmation il doit exister au moins un compilateur pour le traduire. L'écriture du compilateur doit respecter la définition du langage. La définition du langage comporte notamment la description des types de données disponibles, et celle des opérations autorisées sur ces données.

Il faut donc un compilateur pour traduire en langage machine les programmes écrits dans un certain langage. Mais le compilateur est lui-même un programme, alors il a fallu soit l'écrire directement en langage machine, tâche exténuante, soit le compiler, mais pour cela il a fallu un compilateur, ce qui nous renvoie au problème initial. Comment cela a-t-il bien pu se passer ?

Entre le langage machine et les langages compilés il y a un niveau intermédiaire, le langage assembleur. À chaque instruction du langage machine correspond une instruction du langage assembleur, écrite sous forme d'une abréviation alphabétique suivie de quelques opérandes, au lieu du code binaire du langage machine. Il est ainsi possible à un être humain normal d'écrire un programme en langage assembleur, je dirais même que quiconque envisage le métier d'informaticien doit avoir essayé, même si c'est plus laborieux qu'avec des langages d'un niveau d'abstraction plus élevé. Le programme qui traduit le langage assembleur en langage machine se nomme... assembleur, et il est beaucoup plus simple que le compilateur, chargé de traduire un langage plus abstrait [1]. À l'origine des temps, quand il n'y avait que le langage machine et les dispositifs matériels de chargement du programme en mémoire, il fallait écrire en langage machine un micro-assembleur, qui ne pouvait traduire qu'un sous-ensemble du langage assembleur, mais qui permettait d'écrire dans ce langage incomplet un assembleur plus puissant, et après quelques itérations un assembleur complet. En pratique, aujourd'hui, si on veut créer un assembleur pour un nouveau modèle de processeur, on utilise un ordinateur existant et on écrit pour cet ordinateur un assembleur qui génère du langage machine pour le nouveau processeur, et de façon générale on sait écrire des compilateurs qui compilent sur une machine de type A des programmes destinés à s'exécuter sur une machine de type B, complètement différente, cela s'appelle la compilation croisée. Tout cela est possible grâce à l'architecture de von Neumann, qui permet de considérer le texte d'un programme comme des données que l'on peut créer et transformer au moyen d'un autre programme.
[/Chapitre suivant/]
Une question importante en programmation est celle du typage des données : pour faire simple, disons que si nous avons des données de type carotte et des données de type salade, il faudra veiller à ne pas additionner des carottes et des salades. Sauf s'il existe une méthode pour convertir par un procédé chimique inédit des salades en carottes, auquel cas on effectue la conversion, puis on peut faire l'addition.

Certains langages sont laxistes, leur définition n'exige pas que le compilateur vérifie que le programmeur n'a pas additionné des carottes et des salades. C'est au programmeur de savoir ce qu'il fait. Et d'ailleurs ce laxisme peut avoir des avantages : si j'écris un programme d'optimisation du remplissage du coffre de la voiture au retour du marché, et que je sais qu'un kilo de carottes occupe autant de place qu'une salade, additionner salades et carottes sans me soucier de l'éventuelle méthode de conversion des salades en carottes peut satisfaire ma paresse. Mais la plupart du temps ce laxisme engendre des erreurs de programmation, par défaut d'attention du programmeur, parce

que l'enchaînement des opérations peut être compliqué, réparti entre plusieurs sous-programmes (qui sont comme des chapitres du programme d'ensemble) d'un logiciel de grande taille. Une proportion importante des erreurs de programmation sont des erreurs de typage. Une erreur de programmation se manifeste soit par l'arrêt brutal du programme sans produire de résultat, soit par le fait que le programme ne s'arrête jamais (sauf à couper l'alimentation électrique de l'ordinateur), soit par la production d'un résultat faux. Dans tous les cas, s'il s'agit par exemple du programme de pilotage de la fusée Ariane, c'est plutôt embêtant, mais même s'il s'agit du logiciel comptable d'une PME, cela peut être catastrophique pour la PME.

D'autres langages sont moins laxistes, leur définition exige que le compilateur vérifie que le programmeur n'a pas additionné des carottes et des salades. Formulé ainsi cela semble trivial, mais pour un programme constitué d'un grand nombre de sous-programmes avec des données de structures complexes (comme des tableaux à plusieurs dimensions) ce n'est pas évident. Alors, chaque fois que le programmeur aura tenté d'additionner carottes et salades, ou d'appliquer à des salades une opération réservée aux carottes, le compilateur lui enverra un message d'erreur plus ou moins compréhensible, et refusera de traduire le texte de son programme (son *code*) en langage machine. Au premier abord cela rend la programmation plus laborieuse et c'est pénible, mais en définitive cela évite beaucoup d'erreurs à l'exécution. On dit qu'avec de tels langages il est plus difficile d'obtenir un texte accepté par le compilateur, mais qu'une fois que « ça compile » il y a de bonnes chances que le programme soit juste, parce que sa cohérence logique a été vérifiée par le compilateur, dont la vigilance n'est pas limitée par les capacités de concentration finies de l'être humain.

On peut comparer les langages de programmation selon un autre axe, leur style ; de prime abord c'est assez subjectif, une question de goût personnel, mais en fait c'est important parce que du style dépend la lisibilité du texte des programmes. Au cours des années 1970 s'est développé un courant de pensée animé par des gens comme Niklaus Wirth, Edsger Dijkstra, C. A. R. Hoare, issus du groupe de conception du langage Algol, qui prônait la *programmation structurée*, en fait l'application à la programmation du *Discours de la Méthode* de Descartes : « diviser chacune des difficultés que j'examinerais, en autant de parcelles qu'il se pourrait, et qu'il serait requis pour les mieux résoudre ... conduire par ordre mes pensées, en commençant par les objets les plus simples et les plus aisés à connaître, pour monter peu à peu comme par degrés jusqu'à la connaissance des plus composés ». Or une grande leçon de la programmation structurée énonce que la première qualité d'un programme est sa lisibilité, parce que son texte sera lu plus souvent par un humain que par un ordinateur, pour le mettre au point, le corriger ou le modifier.

La lisibilité des programmes dépend pour une bonne part du style du langage de programmation. Certains langages expriment l'articulation des différentes opérations du programme par des mots, généralement empruntés à l'anglais, c'est la famille Algol, Pascal, PL/1, Ada... D'autres langages utilisent pour noter ces articulations des signes cabalistiques tels qu'accolades, barres verticales, éperluettes, astérisques et parenthèses : ce sont C, C++, Perl. Les travaux de didactique de l'informatique ont montré que les langages de la première famille étaient plus lisibles et plus faciles à apprendre que ceux de la seconde. La lisibilité d'un programme ne dépend bien sûr pas seulement du langage de programmation, un mauvais programmeur peut écrire des programmes illisibles avec un bon langage, et un bon programmeur peut écrire de bons programmes avec un mauvais langage, notamment grâce à des commentaires écrits en marge qui permettent de pallier le côté peu expressif [2] du langage.

En 1974 le département de la Défense des États-Unis (DoD) constate que ses équipes informatiques et les entreprises sous-traitantes emploient des dizaines (voire des centaines) de langages différents et que cela lui coûte très cher. En effet, pour chaque nouveau langage les programmeurs sont par définition débutants, la qualité des programmes s'en ressent, les coûts de développement aussi, mais surtout les budgets de maintenance s'envolent au fil des ans et excèdent la dépense initiale par un facteur supérieur à 10. De cette réflexion est issu en 1977 un cahier des charges pour un langage unique, universel et rigoureux, adapté aussi bien à la comptabilité qu'au guidage de missiles balistiques, et qui bannisse absolument l'addition de carottes et de salades, ou, en termes plus techniques, qui soit doté d'un typage des données strict et vérifié. L'appel d'offres subséquent est remporté par une équipe française de CII-Honeywell-Bull dirigée par Jean Ichbiah, avec Véronique Donzeau-Gouge que je rencontrerai plus

tard au Conservatoire national des Arts et Métiers (Cnam). Le langage qui en résulte est nommé Ada, en l'honneur d'Ada Lovelace, fille du poète Byron, sans doute la première programmeuse de l'histoire lors de sa collaboration avec Charles Babbage, inventeur de plusieurs machines à calculer dans la première moitié du XIXe siècle et précurseur de l'informatique moderne.

On l'aura compris, Ada appartient à la famille des langages rigoureux et expressifs. Il est adapté à la réalisation de programmes « temps réel », c'est-à-dire dont on peut garantir la terminaison dans un intervalle de temps borné, fixé à l'avance. Pour le système de pilotage d'un avion, par exemple, les délais se comptent en dizaines de microsecondes. Ada permet également la programmation d'activités concurrentes, c'est-à-dire qui se déroulent simultanément et de manière coordonnée, en échangeant des informations entre elles. Et tout ceci en conservant les propriétés de cohérence qui garantissent la justesse des résultats.

L'Association française pour la cybernétique économique et technique (AFCET), à l'époque la principale société savante francophone pour l'informatique, crée un groupe de travail Ada. Je suis ses réunions avec passion, même si je ne comprends pas toujours la teneur des empoignades homériques mais amicales entre Michel Gauthier de l'université de Limoges et Dominique Chandesris, qui sera plus tard un expert de l'Agence nationale de sécurité des systèmes d'information (ANSSI).

Ada s'imposera pour les applications critiques, dans l'aérospatial, les systèmes de transport tels que la ligne 14 du métro parisien, les installations nucléaires. Il n'aura pas le succès qu'il mérite dans les autres domaines, d'abord parce que sur les ordinateurs des années 1980 le compilateur, complexe du fait de ses ambitions, était lent, ensuite parce qu'il était cher : les compagnies qui en réalisaient visaient le marché militaire et spatial, dont la solvabilité paraissait sans limite. Parmi les langages apparus récemment, Rust, créé par la fondation Mozilla, est un héritier d'Ada par certains aspects, tels que le système de types, les modalités de communication entre sous-programmes et les possibilités de programmation concurrente.

[/Chapitre suivant/]

---

[1] On rencontre ici une loi générale de l'informatique : ce qui est simple pour l'humain est compliqué pour l'ordinateur, et réciproquement. Un langage d'un niveau d'abstraction élevé est plus facile à apprendre et à programmer que l'assembleur, mais son compilateur est beaucoup plus complexe.

[2] Comme nous l'apprend le chercheur Matthias Felleisen, l'expressivité doit être entendue à la fois comme la puissance d'expression, la capacité à exprimer une opération complexe par un formalisme aussi simple que possible, et comme la qualité de l'expression, le fait que le texte du formalisme suggère facilement sa signification au lecteur.