LA TOUR DE BABEL: LE ZERO ET LE FINI

Faut-il un permis de conduire les ordinateurs?

Il n'y a pas si longtemps, il était imprudent de partir avec une automobile si l'on n'était pas capable, au minimum, de "déboucher le carburateur". Aujourd'hui, vous pouvez prendre la route sans même savoir si le moteur est à l'avant de la voiture ou dans le coffre...

En informatique, on constate une évolution un peu analogue. Bien entendu, au niveau professionnel, on ne peut travailler avec un ordinateur sans connaître à fond au moins une partie des possibilités de l'outil; mais toujours plus nombreuses sont les applications notamment en informatique personnelle, qui n'exigent que la maîtrise de quelques manipulations élémentaires. Les constructeurs savent bien qu'une réelle simplification de l'utilisation des ordinateurs est nécessaire pour les rendre aussi banalisés qu'une voiture automobile.

L'année 1982 a vu par exemple le développement à grande échelle de procédés simplificateurs: écrans tactiles, où l'on communique avec la machine en touchant du doigt diverses réponses possibles à un "menu" proposé, et surtout la fameuse "souris", qui répond au même objectif en faisant rouler une boule sur la table. Ces différents substituts ont évidemment pour but de supprimer, dans la mesure du possible, le passage par un clavier, car ce dernier est encore très mal accepté tout particulièrement dans notre pays.

Exécutants passifs et dompteurs de machines

Au niveau le plus bas de la conduite d'une machine, on trouve par exemple les jeux électroniques. Pour vous amuser, nul besoin de savoir lui "parler": les manettes (ou joysticks) le font pour vous en éparpillant vos balles de tennis dans tous les coins de l'écran selon la qualité de vos réflexes.

Au niveau le plus haut, on peut saluer les informaticiens de la NASA qui doivent suivre les évolutions des satellites et leur envoyer à chaque instant les instructions nécessaires - on dit qu'ils travaillent "en temps réel" -. Ils ne peuvent écrire leurs programmes qu'au prix d'une familiarité totale avec les procédures les plus fines de leurs machines.

Au milieu, figure la masse des utilisateurs de terminaux ou d'ordinateurs individuels tournés vers la gestion, qui ont à mettre en oeuvre des programmes professionnels tout prêts (des progiciels). Ils ont parfois besoin d'intervenir eux-mêmes pour écrire de petites applications personnalisées demandant un traitement spécial.

Bien entendu, quelle que soit la part réelle d'intervention de l'homme devant la machine, celle-ci ne peut rendre les services attendus que si elle a reçu toutes instructions pour cela; ce sont les langages informatiques qui le permettent. Simplement, selon le cas, c'est le professionnel qui pilote directement son ordinateur, ou celui qui a tout écrit d'avance (pour que l'amateur de jeux n'ait plus qu'à triturer ses manettes) qui ont à traduire les intentions en procédures compréhensibles par les transistors des "bécanes" - mot affectueux qu'emploient les mordus pour désigner leurs machines -.

Zéro, un: tout un monde informatique

Un livre sérieux sur l'informatique commençait jadis rituellement par une description des beautés des 0 et des 1: ces deux chiffres symbolisent si bien l'univers des ordinateurs qu'ils constituent le nom d'un des journaux les plus connus du secteur! C'est vrai qu'en définitive c'est à coups de zéros et de uns que marche n'importe quel matériel, mais il n'est plus nécessaire aujourd'hui d'infliger au profane la technique, même élémentaire, du calcul binaire: la totalité des amateurs, une grande majorité des utilisateurs professionnels de l'informatique se passent très bien de toutes ces choses s'ils le désirent. Donc pas de cours d'arithmétique modulo 2, de longues explications sur les bits de parités et autres splendeurs...

Sans nous y arrêter plus qu'il n'est nécessaire, il est pourtant utile de donner un très bref coup d'oeil sur ce qu'est, dans la réalité des choses, un programme informatique. Les quelques lignes suivantes constituent un exemple authentique; voilà l'une des possibles nourritures que l'on peut donner au micro-processeur Z80, l'un des plus répandus au début de l'histoire de la micro-informatique, pour lui intimer l'ordre de lire une liste de nombres déjà en mémoire et d'en extraire le plus grand:

```
00100001 10100011 01110010 01000110 10010111 00100011
```

10111110 00110000 00000001 01111110 00010000 11111001
00110010 11110010 01111010

Une sévère discipline: le langage machine.

La simple frappe de ces quelques 120 chiffres est un exercice plutôt désagréable, plein de risques d'erreurs. C'est pourtant la seule forme possible pour atteindre le vrai niveau de la réalité physique; cette chaîne donne exactement la liste des états des mémoires élémentaires du Z80 - certaines "activées", repérées par des 1, les autres "au repos", comme une lampe doit être allumée ou éteinte - pour qu'il puisse faire le travail attendu. Ces listes de 0 et de 1 ont été le pain quotidien des pionniers; elles constituent un exemple (ultra bref, de vraies applications exigent des pages entières) de ce que l'on nomme le "langage-machine", expression qui parle d'elle-même.

Bien entendu le programmeur peut aujourd'hui soulager son travail en substituant, à ces litanies monotones, les nombres décimaux ordinaires dont les chaînes ci-dessus constituent les écritures binaires; on remplacerait donc avantageusement le paragraphe entier par la ligne suivante:

33 163 114 70 151 35 190 48 1 126 16 249 50 242 122

d'écriture plus simple, moins sujette à erreur de frappe, plus facile à contrôler. En fait, c'est encore un autre système qui est utilisé (l'hexadécimal, ou système à base 16 et non 10), pour certaines raisons particulières sans grande importance ici; le "vrai" listing du programme est donc:

21 A3 72 46 97 23 BE 30 01 7E 10 F9 32 F2 7A

ce qui a peu de chance d'apparaître comme un progrès substantiel aux profanes! Mais même si ces procédures sont un petit peu moins pénibles à lire que la toute première, on comprend facilement que travailler avec un tel matériau est plus proche du bagne que de toute autre activité humaine.

Souffrir un peu moins grâce à l'Assembleur

Le langage-machine, par lequel il a bien fallu commencer - ne serait-ce que pour que se construisent les premiers calculateurs permettant le passage automatique des langages plus "évolués" aux 0 et aux 1 -, est maintenant remplacé par ce que l'on appelle des Assembleurs. Ce ne sont pas des machines, mais des programmes, c'est-à-dire des suites d'instructions, exécutables par l'ordinateur lui-même, transformant des ordres un peu moins hermétiques en ces fameux codes binaires 00100001 10100011 ... Ces ordres sont communiqués au programme assembleur par l'intermédiaire de mots d'un langage lui-même appelé assembleur; par exemple ils sont transmis par la frappe au clavier, ou par la lecture d'une bande magnétique, d'un disque ou tout autre moyen d'entrer en communication avec le coeur de la machine (on parle de périphérique d'entrée). Les mots du langage sont connus sous l'appellation de "mnémoniques"; ils sont, très vaguement, inspirés de l'anglais. Voici l'équivalent "assembleur" du tout petit programme en question:

```
LD HL,72A3H
LD B,(HL)
SUB A
L1: INC HL
CP (HL)
JR NC,L2
LD A,(HL)
L2: DJNZ L1
LD (7AF2H),A
```

Une traduction approximative

Le mot LD (en fait, deux lettres) vient du verbe LOAD, qui signifie charger, plus précisément ici: mettre en mémoire; les lettres HL désignent justement le nom de la mémoire spéciale, dénommée registre, où va avoir lieu le stockage annoncé. L'opération LD HL (placer quelque chose dans le registre HL) est relativement parlante au programmeur devant son clavier; elle sera transformée automatiquement en 00100001, c'est-à-dire 33 (en décimal) ou 21 (en

héxadécimal); elle correspond donc bien à la première instruction du programme, mais sous une forme un peu moins hermétique.

Une fois décrit le type de l'opération à exécuter, il faut dire ce qu'il y a à charger dans la mémoire HL: c'est ici un nombre, symbolisé par les signes 72A3H, donc celui qui, en système héxadécimal - noter la présence du H -, s'écrit 72A3, c'est-à-dire enfin l'entier (ordinaire) 29347. Si nous nous reportons à la liste ci-dessus, on voit en effet qu'y figuraient bien les signes 72A3, mais sous la forme... A3 72: pour des raisons qui importent peu, l'usage s'est établi, comme on le voit, d'écrire d'abord... les deux derniers chiffres du nombre avant les deux premiers! (C'est comme çà qu'on a réussi à persuader tout le monde, pendant des années, que l'informatique était réservée à des esprits hors du commun, justifiant des salaires fabuleux).

Voilà donc (plus ou moins expliqué) comment LD HL,72A3H peut devenir 00100001 10100011 01110010. Pour vérifier que le message a été frappé sans erreur et bien compris, le programme assembleur imprimera automatiquement les deux bouts de la chaîne: ce qu'on lui a envoyé (ici LD HL,72A3H), et le résultat de la traduction (sous la forme 21 A3 72, plus facile à contrôler que 00100001 etc...). Resteraient à commenter les ordres SUB (soustraire), INC (incrémenter, ou encore augmenter, le contenu d'un registre), JR (jump relative, c'est-à-dire sauter un certain nombre d'instructions), etc.: une pleine page n'y suffirait pas. Il faudrait également expliquer le vocabulaire plus qu'étrange des professionnels pour qui les assembleurs, à partir du "code-source" (c'est-à-dire des instructions comme LD, mots de code en effet), "génèrent" (c'est le terme consacré) un "code-objet" comme 00100001: voilà qui fait terriblement société secrète....

De la calculette programmable aux langages évolués

Un exposé trop technique importunerait d'ailleurs la plupart des lecteurs de ce livre qui n'auront sans doute jamais à se soucier de ce genre d'exercice de style; la partie visible de la grande majorité des programmes effectivement écrits ou recopiés par les utilisateurs ne ressemble heureusement en rien à ces hiéroglyphes. Notons cependant que les possesseurs des calculatrices programmables les plus simples connaissent bien une sorte de langage assembleur, celui qui contient

les ordres STO (store: emmagasiner, analogue à LOAD), RCL (recall: rappeler) puisque travailler au niveau même des mémoires se traduit par des ordres très précis du genre: mettre tel nombre dans telle mémoire (nommée par son numéro), à la place des instructions beaucoup plus symboliques qu'utilisent les autres langages.

Au delà des langages machine et assembleur, très étroitement soumis à l'architecture physique de l'ordinateur utilisé, il a fallu en effet bien vite mettre au point d'autres techniques de programmation, plus proches des langages naturels - en tous cas des habitudes et notations mathématiques en ce qui concerne les calculs -, plus indépendantes du matériel et, si possible, "portables" sans problèmes: cela signifie que l'on aimerait évidemment pouvoir traduire sans effort tout programme écrit pour la machine XYZ en programme pour sa concurrente UVW. Ces langages sont dits "évolués"; ils sont innombrables.

Enfin FORTRAN vint...

Le tout premier à atteindre ce "haut niveau" est encore très utilisé, compte-tenu de la masse énorme de programmes qu'il a servi à écrire; c'est le célèbre FORTRAN (FORmula TRANslator, traducteur de formules). La possibilité même de créer des langages universels (qualificatif certes trop ambitieux, mais très évocateur) n'était nullement évidente en Décembre 1953 quand John BACKUS, déjà auteur de l'un des langages d'assemblage d'IBM, commença à mettre au point la première version de FORTRAN pour le compte de la même compagnie. Le 10 Novembre 1954 il présentait le nouveau-né, fruit des efforts de 25 hommes-années comme l'on dit dans cette corporation où, décidemment, on se saurait se résigner à parler comme tout le monde.

Le résultat de ce travail est assez difficile à imaginer pour un profane; c'était d'abord un rapport, plus précisément une sorte de grammaire/dictionnaire, où l'on définissait théoriquement les mots du langage, leurs connexions possibles, bref quelque chose comme un manuel d'Esperanto proche des mathématiques où une construction totalement abstraite est exposée telle que l'ont conçue ses géniteurs. C'est donc un travail intellectuel intéressant, mais a priori de difficulté normale. Pourtant l'accouchement de FORTRAN constituait un véritable exploit, comparable à la mise au point de la

fusée Saturne ou la démonstration d'une conjecture mathématique vieille d'un siècle. En effet cet exposé théorique était assez précis pour permettre de réaliser presque aussitôt un compilateur, prouvant ainsi que les buts du rapport n'étaient nullement utopiques mais avaient été réellement atteints.

Des variables libres de choisir leur domicile

Quels étaient les buts de John Backus? Les informaticiens étaient déjà heureux d'avoir pu remplacer l'horrible langage-machine par un jargon un peu moins insipide, automatiquement traduit en 0/1 par les programmes assembleurs. Mais on voulait leur offrir encore mieux: un langage réellement évolué, bien plus lisible à l'oeil nu, qui ne s'attacherait plus à appeler les mémoires par leurs adresses explicites mais pourrait aller à l'essentiel. Les ordinateurs, machines serviles par excellence, peuvent très bien gérer elles-mêmes tous ces problèmes de numéros d'ordre des mémoires: peu importe au programmeur l'endroit précis où transite telle information à tel moment, le système décidant de son propre chef des différents dispatchings, retenant le temps qu'il faut les adresses utilisées etc. Les arbres ne cachant plus la forêt, l'homme peut consacrer son temps précieux (car coûteux) aux vrais problèmes que soulève son travail: le chimiste peut ne penser qu'à la chimie, le comptable à la comptabilité etc. Pour calculer une somme, le programmeur écrit aujourd'hui, comme tout le monde, S = A + B, au lieu de charger telle mémoire avec A, d'envoyer le résultat S dans tel coin des circuits etc.

Pour prouver que ce n'était pas un rêve, Backus et ses collaborateurs avaient donc écrit (en Assembleur, bien sûr) un programme qui traduisait automatiquement les instructions comme S = A + B en code-objet avec des 0 et des 1. Réaliser concrétement une affirmation d'ordre intellectuel est plutôt rare; par exemple, lorsqu'un universitaire rédige une thèse pour démontrer l'existence, jusque-là hypothétique, de tel Pharaon obscur, on ne s'attend pas en général à ce qu'il fournisse en plus la tête de la momie; le commando FORTRAN, lui, avait jeté sur la table du jury la preuve réelle des affirmations contenues dans l'exposé préliminaire. (Par comparaison, plus de cinq ans sépareront le rapp... sur le langage ADA, il est vrai

bien plus complexe, de l'écriture du premier compilateur ADA: ce genre de sport est plutôt éprouvant).

La galaxie FORTRAN: une grande famille

Comme toute entreprise humaine, FORTRAN n'était pas sans défauts; en 1977 encore paraissait une version améliorée, appelée FORTRAN V, preuve que les quatre premières posaient problèmes - on ne pouvait y manier commodément les chaînes de caractères (les lettres constituant les mots) -. Quoi qu'il en soit, la suprématie de ce langage, évidemment fortement liée au succès commercial d'IBM mais utilisé par tous les constructeurs, fut longtemps évidente, surtout pour les applications scientifiques. Mais ce n'était que le début d'un foisonnement intense. A côté de FORTRAN se sont développés, gardant toujours plus ou moins une marque indélébile de filiation, d'autres idiomes spécialisés, par exemple dans les applications industrielles (guidage des machines-outils), ou en gestion, ou en intelligence artificielle, mais aussi des langages encore plus ambitieux que FORTRAN par leur souci de généralité.

Pendant un assez long temps en effet, les ordinateurs et, par conséquent, les programmes associés, semblaient devoir être répartis en différentes familles sans grand lien entre elles. Il semblait évident qu'existaient une informatique de gestion, une informatique scientifique etc. Mais peu à peu on revenait au caractère universel des ordinateurs, en demandant par conséquent des langages également universels. Par exemple, IBM consacra beaucoup d'efforts à la mise au point d'un New Programming Langage, devenu depuis PL/I (Programming Language N° 1), conçu en 1964, réalisé en 1966. Le PL/I cherche à englober l'ancêtre FORTRAN, ainsi qu'ALGOL, l'un de ses concurrents (dont nous dirons un mot plus loin) et surtout COBOL, champion des applications à la gestion. Même si son succès fut réel, il n'a pas réussi à être tout-à-fait aussi polyvalent que ses auteurs l'avaient espéré, et la course à la bonne-à-tout faire n'en fut pas pour autant terminée.

Factures, stocks, bilans... l'univers de COBOL

On ne peut pas, même dans une très brève revue des différents

langages informatiques, passer sous silence le rôle de COBOL (COmmon Business Oriented Langage, 1960). En effet COBOL a connu un développement immense, il a été utilisé dans pratiquement toutes les applications à la gestion, et est resté presque sans rival jusqu'à l'arrivée de PL/I qui ne l'a nullement éliminé. De plus son histoire est assez curieuse pour être contée. Pour la première fois, une Administration joua volontairement un rôle considérable dans l'aventure de l'informatique; avec l'aide d'un groupe d'utilisateurs, le Département de la Défense (le Ministère de la Guerre américain) a mis au point ce langage, indépendant - en principe - des configurations des différents ordinateurs, pour traiter les tâches courantes en gestion. Voulant assurer le succés de l'entreprise, les services gouvernementaux avaient prévenu qu'ils ne signeraient aucun contrat avec une compagnie n'ayant pas de compilateur COBOL sur ses machines... Voilà de quoi faire réfléchir, même si le produit n'était pas vraiment excellent. Détail pittoresque, le principal acteur de l'écriture de COBOL fut Grace HOPPER, née en 1907, pionnière en programmation et, comme son nom ne l'indique pas, officier de la Marine américaine (en 1984, d'après Time, le Commodore Hopper, poing virilement sur la hanche, sanglée dans un magnifique uniforme, est toujours en activité...).

ALGOL ou la volonté de rigueur

Grand-Papa FORTRAN présente encore aujourd'hui quelques défauts, provenant des conditions mêmes de sa naissance puisqu'il constituait pratiquement une première absolue. Ces défauts étaient encore plus évidents dans son adolescence. C'est pourquoi des universitaires, particulièrement européens, voulurent très vite mettre au point un langage conçu, dès le départ, de façon plus axiomatique, avec une grammaire plus rigoureuse. C'est ainsi qu'est né ALGOL - c'est-à-dire ALGOrithmic Language -, défini d'abord à Zürich en 1958, revu en 1960 et encore en 1968, oeuvre de plusieurs groupes réunissant par exemple John BACKUS, père de FORTRAN, l'allemand F.L.BAUER, le français Bernard VAUQUOIS, tous théoriciens et utilisateurs de l'informatique. Si la vie d'ALGOL fut relativement brève, faute de développements industriels, son influence reste importante. La tradition universitaire en fit grand cas, notamment à

cause des notions alors nouvelles de récursivité, de "blocs" (outils de base de la programmation dite structurée). La méthode de création d'ALGOL, rapidement devenu un standard pour les applications scientifiques - mais bien peu apte aux besoins de la gestion - servit de modèle aux créateurs de COBOL.

Où l'on retrouve le maître PASCAL

En fait ALGOL est surtout important comme père de PASCAL, défini à Zürich par le mathématicien suisse Niklaus WIRTH en 1968 (premier compilateur deux ans plus tard), révisé en 1973, modifié par l'Université de Californie à San-Diego pour être plus facilement adapté à des micro-ordinateurs - sur lesquels il a commencé une belle carrière quand les coûts de mémoires sont devenus relativement marginaux -. Le nom de PASCAL est évidemment un hommage à l'inventeur de la calculatrice. Il est d'ailleurs souvent pris comme référence prestigieuse: la tour construite en 1983 par IBM-Europe à la Défense lui doit également son nom. Dans une revue reçue pendant la rédaction de ce chapitre, figurent des programmes de construction de courbes mathématiques célèbres, illustrés d'une caricature du philosophe promenant en laisse le "limaçon de Pascal"; mais cette découverte, pour une fois, est dûe à Etienne Pascal, le père! L'histoire des sciences et techniques n'est pas toujours simple.

PASCAL plaît aux scientifiques par son côté rigoureux, par une lourdeur grave assez scolastique, permettant mal les raccourcis subtils et esthétiques, mais assurant une grande sûreté de mise au point et une bonne lisibilité. Il est en effet fréquent qu'un programme aux ellipses fulgurantes, écrit un jour de grande inspiration, ne puisse plus jamais resservir parce que personne, même son auteur, ne sera plus capable de le relire ou même d'en comprendre l'architecture. Pour obtenir une portabilité minimum, une compréhension relativement aisée par tous, un bon langage doit être "documentable", c'est-à-dire permettre les remarques en langue ordinaire, et "structuré". En particulier il doit vous inciter fortement à n'emprunter que de larges avenues (parsemées d'autant de cailloux du Petit Poucet qu'il le faut pour s'y retrouver sans problèmes), et à préférer les grandes symphonies bien rythmées aux menuets canailles; ces derniers ravissent peut-être l'esprit un moment, comme un bon mot, mais

retombent comme autant de fleurs fanées inutilisables. Les modules fragmentant, à la mode cartésienne, les algorithmes en sous-problèmes élémentaires, s'emboîtent avec des "begin" (début) et des "end" (fin) qui constituent autant de points de passage obligés.

Les langages de la famille de PASCAL se veulent exemplaires de rigueur solennelle (janséniste?). De ce point de vue, la référence pascalienne n'est peut-être pas nécessairement la meilleure possible dans la mesure où elle évoque sans doute le vif-argent, l'ironie souveraine, de brillantes cellules grises, mais pas du tout un bulldozer massif, puissant et efficace comme l'école allemande des mathématiques axiomatiques du début du siècle.

Mais placer l'informatique sous le patronage de Pascal est tout-à-fait justifié. Il aurait profité à plein des ordinateurs: imaginons quel surcroît de mordant aurait apporté un traitement de textes à l'auteur des Provinciales! Quelles facilités aurait offert un gestionnaire de fichiers pour reclasser des Pensées aujourd'hui encore bien emmêlées! Si son fantôme hante à l'occasion les environs de la Place des Vosges, Pascal doit bien sourire des avatars modernes et inattendus de sa boîte de cuivre... Situé à égale distance de sa tombe présumée et de la rue Descartes, le club informatique du lycée Henri IV évoque-t-il parfois le face-à-face du 23 Septembre 1647, la démonstration de Roberval et ses étonnantes et lointaines retombées?

De l'influence de la poésie épique anglaise sur le Pentagone

PASCAL lui-même possède une descendance de qualité, qui a fait se côtoyer étrangement, une fois de plus, littérature et ordinateurs. Déjà responsable du comité COBOL, le DOD (Département de la Défense américain) a cherché à nouveau, vers 1976, à remettre de l'ordre dans ses logiciels informatiques en imposant un langage unique à tous ses services pour la fin des années 80. (On comprend en effet facilement quel effet désastreux peut avoir, en cas de conflit, l'incommunicabilité de données provenant de systémes incompatibles!) Le langage sorti vainqueur en 1979 de cet appel d'offres est visiblement fils de PASCAL. Son nom, ADA, rend hommage à une très curieuse femme du siècle dernier. Lady LOVELACE, collaboratrice de Charles BABBAGE qu'elle aida à programmer sa machine analytique,

était en effet mathématicienne mais aussi née Augusta Ada BYRON, fille unique du mariage malheureux d'un poète que l'on ne s'attend guère à rencontrer au coin d'un bois informatique. (A dire vrai en Janvier 1816, un mois après la naissance d'Ada, un an après les noces, avait lieu la séparation complète et définitive de Byron d'avec sa famille: on peut douter de l'influence paternelle sur le premier programmeur de l'histoire!). Voilà une référence littéraire anglaise et non plus française; toutefois, et peut-être est-ce plus important, Jean ICHBIAH, chef de file de l'équipe victorieuse et ingénieur à la Compagnie Bull, a fait ses études de 1960 à 1962... au numéro 5 de la rue Descartes, à un jet de pierre du pilier de Saint-Etienne-du-Mont: Pascal n'est toujours pas loin! S'il est encore bien trop tôt pour préjuger de l'avenir d'ADA, il est bien probable que les langages de demain, certainement fortement structurés, lui ressembleront sur plus d'un point.

Langage universel ou costumes sur mesure?

Bien entendu, quelques pages d'un livre ne peuvent rendre compte de tous les types de langages effectivement présents sur un grand nombre de machines; même au niveau des simples micro-ordinateurs, voire domestiques, fleurissent des noms étranges comme FORTH, LOGO. Chaque éditeur d'informatique possède "son" livre sur LISP, sur APL, sur C. Chaque constructeur vante les possibilités nouvelles qu'apporte le langage standard de ses systèmes. Ce qu'il faut bien voir, c'est qu'il y a perpétuel conflit entre deux tentations:

- la recherche d'un véritable idiome universel (cf la création de PL/I); si elle n'est pas vraiment utopique, cette démarche conduit forcément à définir une structure très complète, donc riche et lourde, exigeant des capacités de mémoire importantes avec une très grande partie des virtualités sans réelle utilisation - un peu comme si on croyait chic de louer les services du London Symphony Orchestra pour sussurrer quelques berceuses à un nouveau-né; cette ambition est plutôt bien vue en Europe;
- le recours, au contraire, à toute une série de langages limités, spécialisés, en fonction des besoins réels de ce que les anglo-saxons appellent le "end-user", l'utilisateur final. Cette attitude serait plus pragmatique et américaine.

Il est vrai qu'un morcellement des tâches peut sembler rétrograde; il ne facilite bien entendu pas la communication, puisque les standardisations deviennent presque impossibles. Chacun se met alors à développer, en fonction de ses besoins propres, telle ou telle version plus ou moins confidentielle... On se retrouve un peu dans le cas d'un pays dont les différentes langues locales sont si nombreuses qu'on est conduit (comme en Inde ou dans certains pays africains aujourd'hui) à des solutions extrêmes, comme l'adoption de l'anglais ou du français pour les documents officiels.

Il faut en fait tenir un peu chacun des deux bouts de la corde: on ne peut prétendre conduire un ordinateur sans avoir, comme version de base, un langage aussi universel que possible, car on ne peut jamais jurer qu'on ne l'utilisera pas pour des applications imprévisibles au jour de l'installation du système; mais il faut être prêt à multiplier les recours à tous les outils particuliers envisageables, quitte parfois à les forger soi-même en cas de nécessité. A chacun sa musette de bricoleur! Bien sûr l'écriture d'un compilateur est-elle peu envisageable par un petit groupe, mais ce n'est pas tout-à-fait utopique, par exemple avec l'aide de PASCAL.

Faut-il collectionner les langages?

Que signifie concrètement la phrase "le langage L est disponible sur la machine M"? Dans le cas le plus simple, il s'agit d'une présence physique dans l'ordinateur, exploitable dès la mise sous tension; pour les micros de bas de gamme, c'est encore trop souvent la seule possibilité. Mais bien entendu, pour des ordinateurs individuels déjà un peu plus riches, et évidemment pour l'informatique professionnelle, un langage c'est tout simplement le contenu d'une mémoire spécialisée que l'on peut acquérir, par exemple des disquettes pour les machines personnelles; si l'on travaille sur un terminal, on doit généralement choisir le langage à utiliser dès l'amorce de chaque dialogue, car les mémoires de l'ordinateur central contiennent généralement les trois ou quatre types les plus courants: COBOL, FORTRAN...

Très concrètement, il n'y a pas, comme nous l'avons déjà vu, d'autre langage que celui des 0 et des 1; aucun ordinateur n'a jamais "parlé" PASCAL ou PL/I. La possibilité d'écrire des programmes PASCAL signifie simplement que ces programmes dits évolués seront

eux-mêmes soumis à un programme spécial présent dans la machine:
- parce que le constructeur l'y a mis dès le départ sous la forme de mémoire morte (ROM) c'est-à-dire indélébile,
- ou parce que l'utilisateur a déclenché le recours à une mémoire externe, par exemple en insérant une disquette ou en lisant au préalable le contenu d'une bande magnétique.

Personnaliser son système

Comme nous l'avons vu, seule la présence du programme particulier qu'est un compilateur a permis à FORTRAN de soulager la tâche formidable des informaticiens. Aujourd'hui, même une machine personnelle comme le Tandy TRS 80 sur lequel j'écris ce livre permet d'accéder à de très nombreux langages; outre le BASIC interne accolé, pourrait-on dire, aux portes mêmes du micro-processeur, je dispose, grâce à des disquettes, d'un BASIC enrichi, d'un FORTRAN, de deux PASCAL, d'un APL, etc.; cette liste n'est bornée que par mes centres d'interêt, les limites des catalogues des sociétés de logiciel et celles de mes possibilités financières... Je n'aurais garde d'oublier un Assembleur qui, même s'il est tout proche du langage-machine, exige lui aussi son propre traducteur pour finir, comme tout le monde, en salami de 1 et de 0. Les frontières ne sont d'ailleurs pas toujours étanches: de nombreux programmes écrits en langage évolué utilisent des parties rédigées en Assembleur car ces "routines" en accélérent, parfois notablement, l'éxécution.

On voit donc l'extrême simplicité de l'ajout d'un langage à une bibliothèque déjà existante. C'est plus facile que de courir aux "Langues-O" pour suivre les cours d'Ourdou: on achète une mémoire (dans ce cas sous forme de disquettes), le manuel correspondant, un ou deux livres pour en savoir un peu plus le cas échéant ou pour trouver des listings de programmes tout faits, un carnet pour rassembler les remarques personnelles, les "trucs" glanés ici ou là, les corrections des erreurs ou des ambigüités (il y en a toujours) des modes d'emploi officiels. Comme pour les langues humaines, ce sont les premiers pas qui coûtent. Si vous connaissez bien un langage et savez en baragouiner un ou deux autres, l'accès au reste de la bibliothèque est relativement facile car tous ont en commun de nombreuses procédures.

Il y a des exceptions notables, comme le LISP de John Mc CARTHY, par exemple, qui semble incompréhensible au premier abord, comme lorsque l'on essaie en vain ses quelques souvenirs de grec, d'anglais et d'allemand devant une plaque commémorative écrite en hongrois. C'est dû à la nature même des applications de LISP, conçu en 1958 au célèbre Massachussets Institue of Technology, spécialisé en intelligence artificielle; il traite presque exclusivement des expressions symboliques et non des nombres ou des mots comme la plupart des langages usuels. Mais en général on peut se sentir au moins vaguement en pays de connaissance, même devant des cas très particuliers comme celui du langage TEX/WEB, écrit par D. KNUTH pour l'édition de textes mathématiques de qualité typographique (on y sent parfaitement la filiation assumée avec PASCAL) ou les "langages-auteurs" utilisés en EAO (Enseignement Assisté par Ordinateur) pour faciliter la tâche des rédacteurs des programmes interactifs d'éducation.

Un produit populaire: le BASIC

Désigner le langage le plus répandu en informatique n'a pas grand sens tant les problèmes sont différents d'un bout à l'autre de cet univers. Pour le grand public en tous cas, celui qui est le plus connu est bien évidemment le BASIC. Le jeu de mots sur lequel repose son nom est fort classique: en anglais, bien sûr, ces lettres sont les initiales de Beginners All-purpose Symbolic Instruction Code: langage symbolique universel pour débutants. Les professeurs KEMENY et KURTZ en écrivirent la première version en 1965 au Dartmouth College, où l'on enseigne notamment les mathématiques en vue de l'aide à la décision - donc l'informatique -. Ce campus typiquement Nouvelle-Angleterre (le Vermont est situé au Nord de New-York et jouxte le Canada) occupe presque à lui tout seul la petite ville de Hanover; allant interviewer John KEMENY deux ans plus tard, j'eus quelque peine à trouver le College avant de comprendre qu'il m'entourait de toute part...

L'idée de départ était d'offrir une liste très restreinte (une dizaine) d'instructions rudimentaires, dans un but pédagogique: faciliter l'apprentissage (assez pénible en effet) du FORTRAN. A cette époque, l'exploitation des ordinateurs utilisait souvent les

techniques, relativement récentes, de "temps partagé" (time sharing): une grosse unité centrale traite pratiquement au même moment diverses applications; chacun des terminaux en service donne l'impression d'une maîtrise de la machine entière, en jouant par exemple sur les délais très longs (à l'échelle informatique) nécessaires à l'entrée et la sortie des informations transmises en général par imprimante - l'écran de télévision ne viendra que plus tard -. Installés chacun devant un terminal, les étudiants du Dartmouth College écrivaient, dans leur langage basique, des instructions traduites au fur et à mesure (interprétées, comme on dit) en un jargon accessible à l'unité centrale.

Une carrière déjà longue

D'après R.MOREAU, le BASIC avait peut-être subi l'influence du PAF (Programmeur Automatique de Formules), écrit en France en 1958 par D.STARYNKEWITCH pour la Société d'Electronique et d'Automatisme de F.H.RAYMOND, pionnier bien connu des débuts de l'histoire de l'informatique. (Comme d'habitude, il est bien vu dans notre pays de ne croire que ce qui vient d'ailleurs, et on oublie parfaitement le rôle important qu'ont joué, à plusieurs reprises, des sociétés françaises comme la SEA ou, plus tard, R2E. Il est bien plus facile de croire que tout est né en Amérique. Ne recherchons pas trop les véritables raisons qui ont empêché notre industrie de rentabiliser des percées techniques parfois de tout premier ordre!).

PAF avait un système d'instructions (écrites en français, comme "aller en...") très proche du premier BASIC. Celui-ci connut ensuite de nombreux avatars: il a éclaté aujourd'hui en une multitude de dialectes - dont des traductions françaises appelées par exemple Basicois -. S'il n'y a pas de BASIC standard au sens propre tant les extensions proposées sont variées, l'initiative d'une société de logiciels, Microsoft (Bill GATES et Paul ALLEN), qui sut écrire une version assez remarquable pour intéresser les principaux constructeurs, a plus ou moins conduit à une stabilisation des principaux concepts autour des BASIC "Microsoft" - eux-mêmes un peu instables au cours du temps! -. De toutes façons, son origine et son histoire mêmes rendraient peu utile le travail considérable qu'exigerait une structuration complète du langage. La relative

pauvreté de ses instructions fait d'ailleurs que le temps d'adaptation d'un BASIC à un autre est heureusement bref.

BASIC = ordinateur individuel?

Son immense popularité, presque accidentelle, provient naturellement de l'explosion de la micro-informatique depuis 1976. Conçu dans un but d'enseignement, il va a l'essentiel quitte, au moins au bas de la gamme, à perdre en puissance ce qu'il gagne en simplicité: en résultent encombrement limité en mémoire et abaissement du coût. De surcroît, le BASIC courant n'est pas un programme compilateur, mais un interpréteur, beaucoup plus facile à écrire et relativement court. Rappelons que cela veut dire que les instructions ne sont pas transformées en langage-machine une fois pour toutes dès l'entrée du programme, mais au fur à mesure que les lignes défilent (inconvénient: tout est à recommencer à chaque fois, ce qui allonge considérablement le temps d'exécution; avantage: le programmeur peut modifier à chaque instant son texte-source, essayer l'effet du programme petit bout par petit bout).

Le BASIC est lent, ce qui n'a pas de conséquences aussi néfastes chez l'amateur (jeux exceptés) qu'en ambiance professionnelle, mais aussi simple, peu coûteux et interactif; il est donc vite devenu synonyme de micro-ordinateurs. Ce n'est qu'en partie exact, car on le trouve parfois sur des machines bien plus importantes, et on a vu que bien d'autres langages sont maintenant disponibles sur des modèles individuels. Bien entendu, il soulève l'ire de nnombreux informaticiens professionnels, qu'ils en voient les défauts criants ou, peut-être plus perfidement, qu'ils soient choqués de voir un jargon d'amateurs accomplir des tâches théoriquement réservées à de plus hautes techniques.

Une succession ouverte? Le cas LSE

Il est bien vrai que la carrière de BASIC n'est pas nécessairement devant lui. Ses performances médiocres en matière de gestion de fichiers, en calcul scientifique (une précision de six chiffres significatifs est très inférieure à celle de n'importe quelle calculatrice), sa syntaxe dangereuse car trop laxiste le conduiront

peut-être à s'effacer devant une version simplifiée de PASCAL ou un prétendant encore inconnu. Avant de lui faire céder la place, il faudra cependant s'assurer que le successeur présente la même facilité d'apprentissage, un compromis aussi heureux entre la brièveté et le confort d'écriture, une souplesse d'acrobate se logeant dans des mémoires exigües. Et encore, de même qu'en politique dire "nous ne pouvons pas faire pire que nos prédécesseurs" ne suffit pas pour que cela vaille la peine de changer de gouvernement, tout challenger devrait encore prouver qu'il est vraiment le meilleur pour devenir le plus populaire.

En France tout particulièrement le BASIC a été très controversé, en particulier parce nous avons longtemps disposé d'un concurrent tout-à-fait estimable, écrit directement en français à l'Ecole Supérieure d'Electricité, par un groupe sous la direction de J.HEBENSTREIT. Le Langage Symbolique d'Enseignement, ou LSE, s'était inspiré de PAF et, surtout, d'ALGOL sans doute plus que de BASIC. Sur ce dernier, il présente plusieurs avantages, dont la récursivité (déjà l'apanage de PASCAL). La faveur des services publics, notamment de l'Education Nationale, a permis un certain développement de ce langage. Il est aujourd'hui encore parlé par un bon nombre d'enseignants de toutes disciplines. Mais diverses raisons firent que les livres consacrés à LSE furent plus nombreux que les machines qui l'acceptaient... C'est essentiellement à un échec commercial - même des micro-ordinateurs français grand public comme le Thomson TO7 (1983) ne parlent pas LSE - que l'on doit attribuer la responsabilité d'un déclin inéluctable. En outre PASCAL, autre fils d'ALGOL, corrige mieux que LSE certains défauts de BASIC.

La théorie des langages de programmation

Les opérations gérées par des langages comme FORTRAN, PASCAL ou BASIC sont évidemment assez variées: elles peuvent comporter des résultats graphiques - par exemple tracer, sur un écran ou sur une feuille de papier, une droite allant d'un point à un autre -, exécuter des calculs mathématiques sur des nombres, travailler sur des symboles littéraux (trier alphabétiquement une liste de clients...) ou même, par le biais d'organes de sortie spéciaux, mettre en route des appareils électriques à une heure donnée etc. Mais toutes ces actions

élémentaires dont l'étude constitue la théorie de la programmation, objet scientifique tout-à-fait digne d'étude et assez proche des mathématiques et de la linguistique formelle.

Après avoir balbutié les instructions les plus immédiates d'un langage, après avoir recopié ou écrit de petits programmes bien naïfs mais formateurs, le débutant en informatique risque de prendre de bien mauvaises habitudes, qui deviennent vite néfastes devant des applications de taille moyenne ou importante. Il importe donc, s'il veut dépasser si peu que ce soit le pianotage distingué, qu'il se plonge dans les méthodes de construction systématique (voire de vérification) des programmes, bref qu'il entre dans le monde complexe de l'algorithmique.

Pourquoi l'algorithmique?

Ce terme désigne évidemment l'étude des algorithmes, "ensembles de règles précises définissant un procédé (...) destiné à obtenir un résultat déterminé à partir de certaines données initiales" (A.A.MARKOV, cité par P.LE BEUX), "descriptions d'un schéma de comportement exprimées à l'aide d'un répertoire fini d'actions et d'informations élémentaires identifiées, bien comprises et réalisables a priori" (M.LUCAS), "descriptions formelles de procédés permettant, à partir de données et en suivant un plan très précis, d'obtenir un résultat, parfois approché" (J.LABORDE, parlant plus particulièrement de calculs). Cette science classifie, par exemple, les routines standards (par exemple structures répétitives ou boucles, du style TANT QUE....FAIRE...., ou DE I=1 A I=N FAIRE...., et structures alternatives ou tests: SI....FAIRE....SINON....). Elle étudie aussi les "preuves" des programmes.

Ces efforts théoriques de décomposition en modules élémentaires sont indispensables, par exemple, à une bonne structuration - religion dont les grands-prêtres sont, parmi d'autres, le néerlandais Edsger DIJKSTRA (Eindhoven) et le français Jacques ARSAC -. J'emprunte aux "premières leçons de programmation" écrites par ce dernier un conte dans lequel il raille la prétention des programmeurs "expérimentaux" renâclant devant les efforts de théorisation: "Chez les Shadoks, la situation est satisfaisante: les essais de programme continuent à très bien rater. Car c'était un principe de base de la

logique Shadok: ce n'est qu'en essayant continuellement que l'on finit par réussir. Ou, en d'autres termes, plus çà rate, plus on a de chances que çà marche... Leur programme n'était pas très au point, mais ils avaient calculé qu'il avait quand même une chance sur un million de fonctionner... Et ils se dépêchaient de bien rater les 999 999 premiers essais pour être sûrs que le millionième marche."

Entre les joies obscures du fanatique bricolant des routines sophistiquées en langage-assembleur pour gagner 15% du temps d'éxécution de sa version des "Envahisseurs", la boulimie du polyglotte dégustant les subtilités de la syntaxe de son dix-huitième langage de haut niveau et les spéculations théorisantes et/ou terrorisantes de certains structurateurs-modulateurs, on voit que programmation, étude des langages, algorithmique peuvent procurer de nombreuses heures de passion intellectuelle. L'informatique peut être un bon opium du peuple; il en est de pires.