

AVANT-PROPOS

Historiquement, le langage C est le langage naturel du système UNIX. Tous deux ont vu le jour vers le début des années 1970, au sein des laboratoires BELL, aux États-Unis.

Les bases du langage ont été établies par Dennis RITCHIE, à partir du langage B, dû à Ken THOMPSON, l'un des inventeurs d'UNIX. D'ailleurs, le langage B, et son successeur le langage C, furent initialement conçus pour le développement de ce système. Pour cette raison, C est muni d'opérateurs et de constructions permettant de travailler au niveau du codage machine des instructions et des données. En ce sens, il est donc un langage de « *bas niveau* ».

Cependant, ses concepteurs ne se sont pas contentés de créer « *encore un autre langage d'assembleur* ». En effet, C appartient à la famille des langages de programmation structurée, héritant comme ses cousins PASCAL et ADA du langage ALGOL. C'est un langage typé, disposant de fonctions, de structures de données, de structures de contrôle, etc. En ce sens, il est donc un langage de « *haut niveau* ».

Le langage C est initialement développé dans le cadre du projet UNIX sur PDP/7, ordinateur disposant de 32 K mots de mémoire. Une première version est achevée autour de 1973, permettant une réécriture en C de la quasi totalité du système UNIX. Différentes fonctionnalités sont progressivement introduites pour finalement aboutir, en 1978, à la naissance officielle du langage. Ses parents sont Brian KERNIGHAN et Dennis RITCHIE, et l'acte de naissance se présente sous la forme d'un livre d'environ 250 pages intitulé « *The C Programming Language* ». Il connaît un succès quasi immédiat, et se trouve très rapidement disponible sur un grand nombre de machines et de systèmes, y compris sur des micro-ordinateurs. Parmi les multiples raisons de ce succès, on peut citer :

- 1° La simplicité : C est un langage de petite taille et il est assez facile d'assimiler rapidement l'ensemble de ses mécanismes.
- 2° La puissance : C est un langage universel, convenant aussi bien à la programmation système, sa vocation initiale (UNIX en est le meilleur exemple), qu'au codage d'algorithmes et à la représentation de structures de données complexes, au développement d'interfaces ou au calcul numérique.
- 3° La portabilité : les programmes écrits en langage C sont assez facilement portables d'une machine à une autre, y compris sur des systèmes différents.
- 4° La souplesse : le langage n'impose pas de style particulier, et toutes les formes de programmation impérative sont possibles.

- 5° L'efficacité : grâce principalement aux pointeurs et à la variété de ses expressions, le langage C permet d'optimiser « à la main » les sources de programmes.

Bien entendu, la portabilité n'est plus aujourd'hui l'apanage du langage C, et l'optimisation des programmes n'est plus faite « à la main ». Mais au début des années 80, toutes ces caractéristiques étaient à la fois importantes et originales pour un langage de programmation.

L'année 1983 est une nouvelle date importante dans l'histoire du langage. Il a maintenant 5 ans, et il entre dans l'adolescence des langages de programmation. Cela se traduit par la constitution, au sein de l'Institut National Américain de Normalisation — « *American National Standards Institute* », ou ANSI —, du groupe de travail *X3J11* chargé de définir et d'adopter une norme du langage C.

Ce travail aboutit en 1989, avec la parution de la norme ANSI/X3.159-1989. L'événement s'accompagne de la publication de la seconde édition du livre « *The C Programming Language* »¹. L'année suivante, la norme américaine est adoptée par le comité mixte ISO/IEC JTC 1, composé de comités techniques de l'ISO (« *International Organization for Standardization* ») et de l'IEC (« *International Electrotechnical Commission* »). Elle devient norme internationale sous l'appellation ISO/IEC 9899. Le langage C s'appelle désormais « *Langage C standard* » : il est adulte.

Un apport très important de ce travail de normalisation est la spécification de la bibliothèque standard du langage C, c'est-à-dire de l'ensemble des fonctions devant être intégrées à une implémentation revendiquant le label ISO. Ces fonctions étaient, pour une grande part, déjà présentes dans les implémentations traditionnelles, mais leur mise en œuvre pouvait varier sensiblement d'un environnement à l'autre.

Une seconde version de la norme du langage C paraît en 1999, venant combler un certain nombre de lacunes, notamment afin de permettre de rivaliser définitivement avec le langage FORTRAN dans le domaine du calcul scientifique. Il ne s'agit pas d'une simple « *mise à jour cosmétique* » mais d'une modification en profondeur, avec notamment l'adjonction de nouveaux types, l'extension des mécanismes de déclaration ou le contrôle fin de l'arithmétique entière et flottante. Cette nouvelle mouture du langage, référencée sous le nom de C99, mettra d'ailleurs plus de cinq ans à être complètement implémentée dans les compilateurs.

Nous avons souligné que l'histoire du langage C est intimement liée à celle du système UNIX, à tel point que l'ouvrage de référence de KERNIGHAN et RITCHIE comporte un chapitre sur l'interface avec ce système. Revenons donc à la fin des années 80, la période des premiers travaux de normalisation du langage C, mais aussi celle où plusieurs groupes travaillent à la normalisation d'interfaces avec UNIX. Ce projet est connu sous le nom de POSIX — « *Portable Operating System Interface for uniX* » — et a couvert environ une trentaine de sous-projets différents. Parmi ceux-là :

- 1° La norme ISO/IEC 9945-1, ou POSIX.1, adoptée en 1990 et étendue en 1996, décrit une interface de programmation reproduisant un ensemble portable de fonctionnalités du système UNIX : création de nouveaux

¹Brian W. Kernighan et Dennis M. Ritchie, *Le langage C - Norme ANSI*, Dunod 2004, pour l'édition en français.

processus, gestion de l'arborescence du système de fichiers, communications, etc. Elle constitue ce que l'on appelle encore l'API POSIX.

- 2° La norme ISO/IEC9945-2, ou POSIX.2, adoptée en 1993, est consacrée aux commandes, à l'interprète de commandes, et à diverses bibliothèques et logiciels de développement d'applications.

Ces versions ont connu diverses révisions, dont la dernière en date, parue en 2003 (ISO/IEC9945:2003) est organisée en quatre parties : ISO/IEC9945-1:2003 (définitions de base), ISO/IEC9945-2:2003 (interface), ISO/IEC9945-3:2003 (interprète de commande SHELL et utilitaires) et ISO/IEC9945-4:2003 (*rationale*). La norme POSIX dépasse aujourd'hui le cadre du système UNIX et constitue une norme d'interaction avec les systèmes d'exploitation, implémentée sous d'autres environnements, comme par exemple WINDOWS NT.

Mais le panorama que nous venons d'esquisser serait bien incomplet si nous ne mentionnions pas pour finir les projets GNU et LINUX.

Le premier voit le jour en 1984 sous l'impulsion de Richard STALLMAN. Il s'agit d'un projet révolutionnaire de conception d'un système d'exploitation analogue à UNIX, mais totalement libre de droit. Ce projet, développé en langage C, fut le point de départ du *logiciel libre*, dont on perçoit seulement vingt ans plus tard pleinement les apports et les retombées.

Le second, lancé en 1991 par Linus TORVALDS, consiste à recoder en C l'intégralité d'un noyau UNIX, lui aussi libre de droit, en s'inspirant initialement d'un *système jouet* disponible dans le monde académique, le système MINIX.

Ces démarches suscitent rapidement un vif intérêt dans le petit monde des programmeurs. De nombreux développeurs de par le monde rejoignent ces projets et contribuent à leur ascension. Aujourd'hui, le système GNU LINUX, reposant sur les deux projets GNU et LINUX, est distribué sous licence GPL, la licence qui définit et protège le logiciel libre, et constitue une implémentation de système à la UNIX de très grande qualité. Il est fiable, efficace, et parfaitement conforme à la norme POSIX. Très répandu dans le monde académique, le système GNU LINUX est également utilisé dans de nombreuses applications industrielles, et il connaît un succès croissant dans le monde institutionnel et dans celui de l'informatique domestique.

PRÉSENTATION DE LA QUATRIÈME ÉDITION

Venons-en maintenant à l'organisation de cet ouvrage, et plus particulièrement à ce qui a fait l'objet de cette quatrième édition. L'ouvrage comporte huit chapitres, dont un chapitre introductif, cinq chapitres présentant l'intégralité des définitions de la norme C99, un chapitre présentant l'intégralité des définitions non optionnelles de la norme POSIX.1, et un chapitre de méthodologie. Plus précisément :

- le premier chapitre est une introduction à la programmation en langage C ;
- les chapitres 2, 3 et 4 sont consacrés respectivement, aux mécanismes de déclaration, à la construction des expressions, et à celle des instructions ;
- le chapitre 5 décrit les directives du préprocesseur ;
- les chapitres 6 et 7 présentent respectivement les fonctions de la bibliothèque standard C99, et les fonctions d'interface système POSIX.1 ;
- le chapitre 8 traite de la modularisation des programmes C. Il présente une

méthode d'écriture de modules réutilisables, basée sur la programmation par objets.

Nous avons souhaité conserver la démarche pédagogique des premières éditions, en essayant de faire en sorte que le document soit lisible « *dans la continuité* » et utilisable comme un support de cours.

Nous avons également souhaité que cette édition puisse faire office de manuel de référence, c'est-à-dire contenant la totalité des définitions de la norme C99 et de la norme POSIX.1, et dont chaque entrée soit complète et directement utilisable par un programmeur. En particulier, lorsqu'une information se trouve répartie en plusieurs endroits, les nombreux renvois insérés dans le texte doivent permettre d'en retrouver rapidement la globalité.

L'arrivée de la norme C99 a représenté une évolution importante du langage. Pour se faire une idée de l'ampleur du changement, on notera que le document de référence est passé de 220 pages pour la première édition de 1990 à plus de 540 pages pour celle de 1999. Les répercussions de cette évolution sur la nouvelle édition de cet ouvrage sont de deux ordres.

D'une part, certains ajouts et extensions de la norme C99 nous ont amené à adapter différents principes de programmation et à revoir en conséquence l'intégralité des programmes d'exemple. À ce titre, on peut notamment citer :

- la possibilité de mélanger les déclarations et les instructions, et donc de retarder les déclarations afin de limiter la portée des objets déclarés ;
- l'introduction explicite du type booléen ;
- diverses extensions venues enrichir les mécanismes de déclaration, comme les tableaux de taille variable ou les fonctions en ligne.

D'autre part chaque nouveauté a fait l'objet, selon son importance, d'un paragraphe ou d'une section nouvelle. C'est par exemple le cas de :

- l'introduction d'un type complexe et d'une arithmétique associée ;
- l'ajout de nouveaux types entiers, permettant d'effectuer de manière fiable le passage progressif aux architectures 64 bits ;
- l'ajout de différents mécanismes de définition de constantes (caractères universels, constantes flottantes hexadécimales, constantes agrégats, etc.) ;
- l'ajout de nouveaux mécanismes d'initialisation des agrégats, permettant de n'initialiser que certaines parties d'un vecteur ou d'une structure ;
- la spécification d'une arithmétique entière plus fiable, en particulier en ce qui concerne les divisions ;
- la prise en compte de la norme ISO/IEC 60559 spécifiant à la fois le format des flottants (incluant des valeurs spéciales comme les infinis ou la valeur indéfinie), et l'arithmétique flottante (modes d'arrondi, exceptions, etc.) ;
- l'extension des fonctionnalités du préprocesseur, avec l'introduction de pragmas standard, de nouvelles macros prédéfinies, et d'un mécanisme de macrofonction avec nombre variable de paramètres ;
- la possibilité d'optimiser la gestion des pointeurs au moyen d'une spécification de restriction d'accès.

Enfin, les modifications les plus importantes concernent la bibliothèque de fonctions standard dont le nombre, d'environ 150 dans la première version de

la norme, est passé à plus de 480. Parmi ces modifications on peut citer :

- l'introduction de nouvelles fonctionnalités de formatage associées aux nouveaux types ;
- l'adjonction de mécanismes de contrôle de l'environnement de calcul flottant et d'interaction avec le processeur de calcul ;
- l'extension des mécanismes généraux de formatage et d'entrée/sortie ;
- l'adjonction de nouvelles fonctions numériques ;
- la généralisation de l'interface des fonctions numériques flottantes et l'introduction d'un mécanisme de généricité ;
- la généralisation aux caractères larges des fonctions de gestions de caractères, de chaînes de caractères, de formatage et d'entrée/sortie ;
- l'extension des mécanismes de liste variable de paramètres de fonction.

En conclusion, toutes les définitions (types, instructions, macros, constantes, variables et fonctions) de la norme C99 figurent sous une forme ou une autre dans cette quatrième édition, dont la rédaction s'est principalement appuyée sur les quatre documents suivants :

- *Programming Languages — C*. Norme ISO/IEC 9899:1999. Second edition. 1999.
- *Rationale for International Standard -- Programming Language -- C*, <http://wwwold.dkuug.dk/JTC1/SC22/WG14/www/docs/n897.pdf>. Revision 2, 20 October 1999.
- *Information technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) [C Language]*. Norme ISO/IEC 9945-1:1996(E). 1996.
- *The GNU C Library Reference Manual – Version 2.3.x* Free Software Foundation. 2001.

CONVENTIONS TYPOGRAPHIQUES

Voici maintenant les conventions typographiques en usage dans l'ensemble de l'ouvrage :

- 1° Les sigles, les noms déposés, les noms de logiciel sont typographiés en petite capitale : ISO, UNIX, GNU EMACS.
- 2° Un mot ou une locution sont typographiés en caractères gras lors de leur définition, comme dans la phrase : « Cet ensemble de fonctions constitue la **bibliothèque standard** du langage C ».
- 3° Les noms de caractère sont typographiés en petite capitale avec une police de caractères fixe : ESC, CR, SPACE.
- 4° Les littéraux du langage C ou de l'environnement de programmation (constantes, identificateurs de variable, mots-clés, noms de fichier et de commande...) sont typographiés avec une police de caractères fixe penchée : "abcd", *compteur*, *void*, *<stdio.h>*, *gcc*.
- 5° Les méta-variables sont typographiées avec une police de caractères italique grasse : ***chemin₁***, ***ancien-nom***.
- 6° Par souci d'homogénéité avec le source des programmes, les nombres réels sont notés avec un point décimal (3.14) et non une virgule (3,14).

7° Les éléments optionnels d'une définition (paramètres d'une fonction, arguments d'une commande, etc.) sont écrits entre crochets épais (**[]**). Lorsqu'un élément peut être répété plusieurs fois, il est suivi de la notation **...**. Exemple :

```
struct [nom]
{
    unsigned [ident]: longueur;
    ...
}
```

8° Un texte placé entre une marque de début de la forme :

```
===== nom.c =====
```

et une marque de fin de la forme :

```
===== nom.c =====
```

reproduit le contenu du fichier **nom.c**. Le contenu du fichier est directement inséré dans le source du livre par le système de traitement de texte. Il est typographié en utilisant une police fixe : `main (int argc, char *argv[])`. Chacun des fichiers ainsi présentés a été compilé et testé.

9° Les exemples d'exécution de programme sont typographiés avec la même police de caractères que les sources de programme, et signalés par un trait vertical dans la marge :

```
| $ Combien y a t il de mots dans cette phrase
| 10 mots.
| $
```

Pour finir, signalons que la composition du texte a été effectuée au moyen du traitement de texte \LaTeX (version $\text{\LaTeX}2_{\epsilon}$), de l'éditeur GNU EMACS, et de scripts SHELL sous BASH. \LaTeX est un logiciel développé par Leslie LAMPORT à partir du système de traitement de texte \TeX conçu et réalisé par Donald E. KNUTH. Les figures ont été réalisées avec l'éditeur XFIG. Les captures d'écran et l'illustration de couverture ont été effectuées avec le logiciel THE GIMP. Les conversions en POSTSCRIPT ont été effectuées avec les logiciels DVIPS et PBM. Les programmes C ont été développés sous GNU LINUX avec les outils GNU EMACS, GCC et GDB.

Les traitements de texte \TeX et \LaTeX font partie du domaine public. Le système GNU LINUX est les autres outils cités ci-dessus sont des logiciels libres distribués sous licence GPL (*GNU General Public Licence*).

REMERCIEMENTS

J'exprime ma reconnaissance sincère à tous ceux qui ont, d'une manière ou d'une autre, inspiré et soutenu ce travail.

Tout d'abord à Patrick GREUSSAY, sans qui cet ouvrage n'aurait certainement jamais vu le jour. C'est lui qui, au début des années 1980, anima le Vax 11/780 du Gréco Programmation du CNRS sur lequel j'ai découvert le monde UNIX et le langage C. C'est encore lui qui, par ses encouragements, et grâce aux nombreux exemples de programmes qu'il mit à la disposition de la communauté informatique française à une époque où Internet n'avait pas encore tissé sa toile, me permit de devenir un « *Céiste Unoxidable* ». Et c'est surtout lui qui, alors que je venais de terminer la première mouture d'un polycopié d'une trentaine de pages intitulé « *Un peu de C* », me glissa l'idée d'en faire un livre.

Ensuite à Jean BERSTEL et à Robert CORI, qui m'ont donné l'élan nécessaire pour démarrer cette entreprise et m'ont permis « *d'y croire* ».

Mais aussi à tous ceux qui, au fil de ces quinze années et de ces quatre éditions, par leurs conseils, par leurs encouragements, ou par le soin qu'ils ont apporté à la relecture de l'une ou l'autre des versions de cet ouvrage, ont permis qu'il voit le jour et qu'il vive sa vie. À ce titre je voudrais notamment remercier Jean-Jacques BOURDIN, Pierre CASTÉLAN, Serge CHAUMETTE, Viviane DELÉTAGE, Irène DURAND, Georges EYROLLES, Frédéric (*Babb's*) GOUDAL, Pascal GUITTON, Patrick HENRY, Michel PALLARD, François PELLEGRINI, Jean-Guy PENAUD, Pascal VALOIS et Bernard VAUQUELIN, ainsi que Cécile RASTIER, des éditions Masson, pour son aide sur la troisième édition.

Je remercie tout particulièrement Jean-Philippe DOMENGER, Philippe NARBEL et Robert STRANDH pour les échanges multiples et fructueux que nous avons régulièrement depuis des années, et Olivier BAUDON, Pascal DESBARATS, Sylvain MARCHAND et à nouveau Jean-Philippe DOMENGER et Robert STRANDH pour leur contribution précieuse à la correction de cette quatrième édition.

J'exprime également toute ma gratitude à Jean-Luc BLANC et Carole TROCHU, des éditions Dunod, pour leur soutien et leur aide efficace dans la correction et mise en forme du document final.

Merci encore à Éric CORBÉRAND pour sa patience et pour avoir réveillé en moi le plaisir d'écrire, et à Soïg SIBÉRIL, Didier SQUIBAN, Henry TEXIER, Dan AR BRAZ, Roland BECKER, Marc COPLAND, Erik MARCHAND, Jacky et Patrick MOLARD, Marc PERRONE, Tchavolo SCHMITT, Archie SHEP, Mal WALDRON, et Franck ZAPPA dont les musiques généreuses et belles m'ont accompagné et soutenu durant les interminables nuits de rédaction, et bien sûr et toujours, merci à Jean-Michel COUDURIER pour ses remarques pertinentes sur les effets de bord.

Mais ce travail n'aurait jamais pu aboutir sans le soutien, la collaboration et la confiance sans faille de Myriam DESAINTE-CATHERINE, qui m'a consacré un temps ô combien précieux à travers d'innombrables discussions, conseils, remarques, suggestions, critiques, relectures, corrections..., et qui accepte depuis plus de quinze ans, avec une patience sans limite, l'intrusion périodique dans notre vie privée d'un bien encombrant pensionnaire.

Et merci, enfin, à TRISTAN et ALICE pour leur gentillesse, pour leur compréhension, et pour m'avoir laissé travailler de longs mois en toute quiétude.

AVANT-PROPOS DE LA PREMIÈRE ÉDITION — 1990

Personne ne sait vraiment définir ce qu'est le « *swing* » et paradoxalement tout le monde sait le reconnaître. Il en va de même avec le style en programmation : personne ne se risquerait à donner « *la* » définition d'un programme bien écrit, mais tout programmeur expérimenté est en général capable d'apprécier la qualité d'un programme à sa seule lecture. Et très souvent, les avis en la matière convergent.

Plus précisément, deux spécialistes du langage C sont en général d'accord pour encenser tel ou tel style de programmation, on dit parfois paradigme, et rejeter tel autre que, bien évidemment, deux experts du langage ADA revendiquent. Par conséquent, lorsque l'on parlera dans ce livre de « *programme bien écrit* », ou lorsque l'on donnera un « *conseil de programmation* », il faudra comprendre « *programme C bien écrit* », ou « *conseil de programmation en C* ».

Une des caractéristiques fondamentales de C est d'assurer la continuité entre le haut et le bas niveau au sein d'un même formalisme. Nous pensons que c'est ce qui fait son grand intérêt. Seulement, un langage de programmation ne peut privilégier la puissance d'expression et la souplesse d'utilisation, et être dans un même temps l'incarnation d'un style de programmation particulier. C'est la contrepartie : C est un langage à risques, ne véhiculant aucune méthode spécifique.

Le « *style C* » s'est développé peu à peu au sein de la communauté des « *céistes* » de par le monde. Car cette communauté existe ; quotidiennement, ses membres échangent des idées, transmettent des informations, posent des problèmes, proposent des solutions..., grâce à l'irremplaçable outil que constitue le réseau INTERNET qui permet à une station de travail localisée au cœur du Texas de converser avec sa consœur bretonne ou occitane.

Par conséquent, en rédigeant ce livre, nous avons eu pour souci principal de mettre en forme cet ensemble de principes, de techniques et de méthodes qui font maintenant partie de la culture d'un programmeur C averti, et sans lesquels il est très difficile de bien programmer en C. Il comporte évidemment un grand nombre d'exemples de programmes, mais aussi de sessions, réalisées directement sous le système (il s'agit ici du système UNIX), et illustrant le résultat de leur exécution. Nous montrons également divers outils indispensables comme les gestionnaires de programmes, débogueurs symboliques, etc.

Nous avons alors dû faire un choix : soit montrer les logiciels classiques contenus en standard dans les distributions officielles des constructeurs, soit montrer leurs homologues « *officieux* » couramment utilisés par la communauté des programmeurs. Nous avons opté pour la seconde solution en prenant le parti de mettre en avant l'environnement avec lequel nous travaillons quotidiennement, et tout particulièrement les remarquables logiciels du projet GNU.