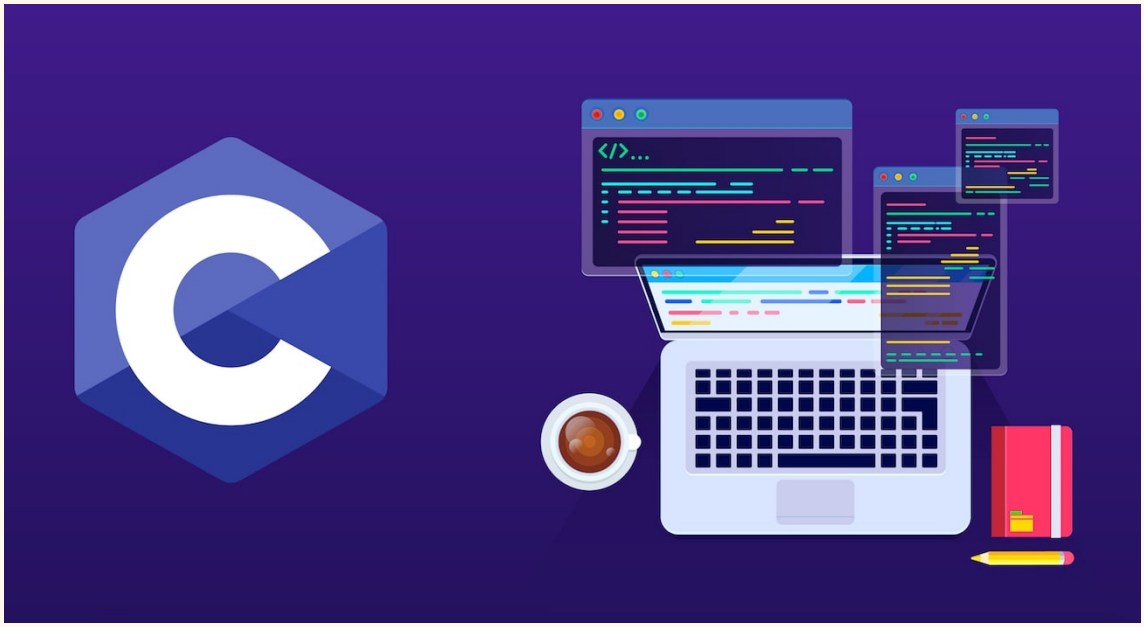
Veille Technologique

LE LANGAGE C
LE PRÉCURSEUR DES LANGAGES MODERNES

—

Par Toutain Jimmy



INTRODUCTION

Cette veille technologique est en lien avec le langage de programmation C. Le C est un langage de programmation impératif généraliste, de bas niveau. Il a été inventé pour réécrire UNIX, une famille de systèmes d'exploitation, dans le début des années 1970. Il est aujourd'hui l'un des langages les plus utilisés, il est le précurseur des langages modernes tels que C++, C#, Java, PHP et JavaScript, ces derniers utilisent la logique du C et une syntaxe similaire. La particularité du C, c'est qu'il permet au développeur, grâce à son bas niveau d'exécution, de pouvoir gérer la mémoire et d'avoir un contrôle important sur la machine. C'est par ailleurs pour cela qu'il est le plus utilisé pour réaliser les fondations (des compilateurs, des interpréteurs, etc...) des langages modernes.

D’où vient t-il ?

Tout d’abord le langage C a été créé avec les bases du langage B, C étant un dérivé de B. C’est Dennis Ritchie qui inventa ce langage, successeur de B, inventé par Kenneth Thompson, qui s'est lui-même inspiré du langage BCPL. Dennis Ritchie à pu faire évoluer le langage B dans une nouvelle version suffisamment différente, en ajoutant notamment les types, pour qu'elle soit appelée C.

Ses évolutions et améliorations

Ce n’est pas parce que le langage C a été créé dans les années 1970 ou qu’il y a maintenant des langages modernes, qu’il ne subit plus de modifications. Il faut savoir que le langage a été souvent amélioré et est normalisé. Les premières ont commencées en 1990 par un comité formé en 1983 par l’ANSI, a abouti à la norme ANSI C en 1989, elle a été adoptée en 1990 par l’ISO (Organisation internationale de normalisation). A ce moment-là le C avait déjà reçu quelques modifications tirées du C++, comme la notion de prototype et les qualificateurs de type. Seulement quatre ans après, entre 1994 et 1996, des publications de correctifs étaient apparues.

Trois fichiers d'entêtes ont été ajoutés, dont deux concernant les caractères larges et un autre définissant un certain nombre de macros en rapport avec la norme de caractères ISO 646.

Cette fois ci, en 1999, une norme nommé C99, les nouveautés portent notamment sur les tableaux de taille variable, les pointeurs restreints, les nombres complexes, les littéraux composés, les déclarations mélangées avec les instructions, les fonctions inline, le support avancé des nombres flottants, et la syntaxe de commentaire de C++. La bibliothèque standard du C a été enrichie de six fichiers d'en-tête depuis la précédente norme.

Une nouvelle fois, en 2011, l’ISO ratifie une nouvelle version standard nommée C11. Cette évolution introduit notamment le support de la programmation multithread, les expressions à type générique, et un meilleur support d'Unicode.

Enfin, encore aujourd'hui une mise à jour est en cours de développement. Toutes ces normalisations montrent à quel point le C est encore en croissance et subit des améliorations afin de correspondre à l'évolution de la technologie.

Ses caractéristiques

Le C est qualifié de langage de bas niveau, ce terme veut signifier que le C est propice à gérer correctement la mémoire en utilisant des entiers et des flottants qui sont conçus pour pouvoir correspondre directement aux types de données supportés par le processeur. Enfin, il fait un usage intensif des calculs d'adresse mémoire avec la notion de pointeur. Un pointeur est une variable directement liée à une adresse mémoire.

Comme le C est un précurseur des langages d'aujourd'hui et qu'il opère à bas niveau, il ne gère de variable complexe, du moins initialement, ces variables complexes peuvent aller de la chaîne de caractères au énumération en passant par les fichiers de données ou même une liste. Cependant cela n'est pas impossible pour le C de gérer ces types de variable plus évolué, pour ça il faut les traiter en manipulant des pointeurs et des types composés.

Par rapport à un de ses dérivés, C#, le C ne propose pas, en standard, la gestion de la programmation orientée objet, ni de système de gestion d'exceptions. Cependant il existe des fonctions standards pour gérer les entrées-sorties et les chaînes de caractères, mais contrairement à d'autres langages, aucun opérateur spécifique pour améliorer l'ergonomie. Ceci peut permettre au développeur d'avoir un contrôle sur son code et de pouvoir fixer les règles de fonctionnement, c'est l'un des bienfaits d'un langage de bas niveau.

Ces caractéristiques en font un langage privilégié quand on cherche à maîtriser les ressources matérielles utilisées, le langage machine et les données binaires générées par les compilateurs étant relativement prévisibles. Ce langage est donc extrêmement utilisé dans des domaines comme la programmation embarquée sur microcontrôleurs, les calculs intensifs, l'écriture de systèmes d'exploitation et les modules où la rapidité de traitement est importante.

En contrepartie, la mise au point de programmes en C, surtout s'ils utilisent des structures de données complexes, est plus difficile qu'avec des langages de plus haut niveau. En effet, dans un souci de performance, le langage C impose à l'utilisateur de programmer certains traitements (libération de la mémoire, vérification de la validité des indices sur les tableaux…) qui sont pris en charge automatiquement dans les langages de haut niveau.

Ses types de variable

Le langage C comprend de nombreux types de nombres entiers, occupant plus ou moins de bits. La taille des types n'est que partiellement standardisée : le standard fixe uniquement une taille minimale et une magnitude minimale. Les magnitudes minimales sont compatibles avec d'autres représentations binaires que le complément à deux, bien que cette représentation soit presque toujours utilisée en pratique. Cette souplesse permet au langage d'être efficacement adapté à des processeurs très variés, mais elle complique la portabilité des programmes écrits en C.

Chaque type entier a une forme « signée » pouvant représenter des nombres négatifs et positifs, et une forme « non signée » ne pouvant représenter que des nombres naturels. Les formes signées et non signées doivent avoir la même taille.

Le type le plus commun est int, il représente le mot machine.

Contrairement à de nombreux autres langages, le type char est un type entier comme un autre, bien qu'il soit généralement utilisé pour représenter les caractères. Sa taille est par définition d'un byte.

Ses bibliothèques

Le langage C étant un des langages les plus utilisés en programmation, de nombreuses bibliothèques ont été créées pour être utilisées avec le C : glib, etc. Fréquemment, lors de

l'invention d'un format de données, une bibliothèque ou un logiciel de référence en C existe pour manipuler le format. C'est le cas pour zlib, libjpeg, libpng, Expat, les décodeurs de référence MPEG, libsocket, etc.

La bibliothèque standard normalisée, disponible avec toutes les implémentations, présente la simplicité liée à un langage bas-niveau. La bibliothèque standard normalisée n'offre aucun support de l'interface graphique, du réseau, des entrées/sorties sur port série ou parallèle, des systèmes temps réel, des processus, ou encore de la gestion avancée des erreurs (comme avec des exceptions structurées). Cela pourrait restreindre d'autant la portabilité pratique des programmes qui ont besoin de faire appel à certaines de ces fonctionnalités, sans l'existence de très nombreuses bibliothèques portables et pallient ce manque.

Sa compilation

Un programme écrit en C est généralement réparti en plusieurs fichiers sources compilés séparément.

Les fichiers sources C sont des fichiers texte, généralement dans le codage des caractères du système hôte. Ils peuvent être écrits avec un simple éditeur de texte. Il existe de nombreux éditeurs, voire des environnements de développement intégrés (IDE), qui ont des fonctions spécifiques pour supporter l'écriture de sources en C.

L'usage est de donner les extensions de nom de fichier **.c** et **.h** aux fichiers source C. Les fichiers **.h** sont appelés fichiers d'en-tête, de l'anglais header. Ils sont conçus pour être inclus au début des fichiers source, et contiennent uniquement des déclarations.

Lorsqu'un fichier **.c** ou **.h** utilise un identificateur déclaré dans un autre fichier **.h**, alors il inclut ce dernier. Le principe généralement appliqué consiste à écrire un fichier **.h** pour chaque fichier **.c**, et à déclarer dans le fichier **.h** tout ce qui est exporté par le fichier **.c**.

La génération d'un exécutable à partir des fichiers sources se fait en plusieurs étapes, qui sont souvent automatisées à l'aide d'outils comme make, SCons, ou bien des outils spécifiques à un environnement de développement intégré. Les étapes menant des sources au fichier exécutable sont au nombre de quatre : précompilation, compilation, assemblage, édition de liens.
Lorsqu'un projet est compilé, seuls les fichiers **.c** font partie de la liste des fichiers à compiler ; les fichiers **.h** sont inclus par les directives du préprocesseur contenues dans les fichiers source.

La phase de compilation consiste généralement en la génération du code assembleur. C'est la phase la plus intensive en traitements. Elle est accomplie par le compilateur proprement dit. Pour chaque unité de compilation, on obtient un fichier en langage d'assemblage.

Cette étape peut être divisée en sous-étapes :

1. **L'analyse lexicale, qui est la reconnaissance des mots clé du langage ;**
2. **L'analyse syntaxique, qui analyse la structure du programme et sa conformité avec la norme ;**
3. **L'optimisation de code ;**
4. **L'écriture d'un code isomorphe à celui de l'assembleur (et parfois du code assembleur lui-même quand cela est demandé en option du compilateur).**

Par abus de langage, on appelle compilation toute la phase de génération d'un fichier exécutable à partir des fichiers sources. Mais c'est seulement une des étapes menant à la création d'un exécutable.

Ensuite l’assemblage, cette étape consiste en la génération d'un fichier objet en langage machine pour chaque fichier de code assembleur. Les fichiers objet sont généralement d’extension .o sur Unix, et .obj avec les outils de développement pour MS-DOS, Microsoft Windows, VMS, CP/M… Cette phase est parfois regroupée avec la précédente par établissement d'un flux de données interne sans passer par des fichiers en langage intermédiaire ou langage d'assemblage. Dans ce cas, le compilateur génère directement un fichier objet.

L'édition des liens est la dernière étape, et a pour but de réunir tous les éléments d'un programme. Les différents fichiers objet sont alors réunis, ainsi que les bibliothèques statiques, pour ne produire qu'un fichier exécutable.

Le but de l'édition de liens est de sélectionner les éléments de code utiles présents dans un ensemble de codes compilés et de bibliothèques, et de résoudre les références mutuelles entre ces différents éléments afin de permettre à ceux-ci de se référencer directement à l'exécution du programme. L'édition des liens échoue si des éléments de code référencés manquent.

CONCLUSION

J’ai décidé de choisir le langage C, pour ma veille technologique, car j’ai toujours apprécié connaître les significations de certains morceaux de code dans des langages de plus haut niveau, donc forcément connaître leur “ancêtre” me permet de mieux comprendre le fonctionnement de certains langages. Je trouve aussi très intéressant de se pencher dans les langages de bas niveau pour apercevoir le mécanisme qui transforme des lignes de code en binaire , 0 ou 1, cela m’a toujours intrigué. Je pense aussi que le C est parfait pour apprendre le codage et le développement car celui-ci nous demande beaucoup de réflexion étant donné son fonctionnement, alors qu’avec des langages de plus haut niveau les solutions sont très souvent déjà implémenter, il ne reste plus qu'à les utiliser.