Julien Crambes

CC1-programmation C

Qu'est-ce que le langage de programmation “C” ?

Table des matières

1 - introduction

Le C est un langage de programmation, donc c'est une langue constituée d'un vocabulaire désignant les différentes **instructions** à donner à un ordinateur. Le C est un langage dit **impératif** car il est constitué de commandes que l'on peut enchaîner les unes à la suites des autres, contrairement à un langage dit **fonctionnel** dont je ne détaillerai pas le fonctionnement ici.

Le C est un langage **interprété**. C'est-à-dire qu'il faut un **compilateur**, un programme qui va transformer les mots en instructions binaires lisibles par le processeur. Cependant, le C a une place particulière parmi les langages interprétés puisque c'est celui qui est “le plus proche de la machine” et donc le plus rapide. Je m'explique : un langage doit être “traduit” pour que l'ordinateur le comprenne. Hors, moins le langage est éloigné de la “langue d'origine de l'ordinateur, plus ce dernier va avoir de facilités pour le comprendre et l'exécuter. Il s'avère que le C est un langage très proche de celui de l'ordinateur ce qui en fait un des langages les plus rapides.

Voici un tableau de différents langages de programmations :

langage	difficulté d'apprentissage	vitesse d'exécution
assembleur	★★★★★	★★★★★
C	★★★★	★★★★
C++	★★★	★★★★
Java	★★	★★★
Python	★	★★

2 - Avantages et Inconvénients

Le C est un langage très rapide comparé aux autres langages usuels. C'est en partie parce qu'il est possible de gérer la mémoire utilisée par notre programme de manière très précise grâce aux éléments inclus de base.

Plus précisément, quand je parle de “mémoire”, je fais référence à la **RAM** (Random Access Memory) représentée par une structure dite de **pile** qu'il faut voir comme une *pile* de feuilles : pour voir ce qui est écrit sur une feuille en dessous du paquet il faut retirer celles du dessus. Ainsi, le C nous permet de verrouiller certaines zones de la mémoire afin qu'elles soient réservées pour notre programme. Pour chaque type de données il y a une place prévue dans la mémoire et à chaque élément on associe une **adresse mémoire** afin de pouvoir le retrouver.

Cette adresse est plus couramment appelée **pointeur** et elle est représentée par un nombre en hexadécimal*. C'est ici que se trouve la puissance de calcul du C : au lieu de travailler avec les données on travaille avec les adresses qui sont souvent bien plus légères que les données qu'elles représentent. Typiquement, au lieu de déplacer un tableau de quelques ko d'informations il suffit de travailler avec sa référence qui ne fait que quelques octets.

Toutefois, la liberté offerte sur la gestion de la mémoire est à double tranchant. Une mauvaise gestion de la mémoire peut provoquer des **fuites de mémoire** (memory leaks) où le programme va demander de la mémoire sans jamais la rendre ce qui va saturer la pile système et provoquer une erreur grave : la fameuse erreur *Stack overflow* (= débordement de la pile). Ce genre d'erreur arrive lorsque le programmeur a oublié en fin de programme de libérer la mémoire qu'il a demandée au début.

Pour finir, on notera que contrairement aux autres langages, il faut beaucoup de patience pour commencer à faire fonctionner son premier programme en C à cause de l'environnement de développement élaboré qu'il faut installer avant de vouloir écrire la moindre ligne de code.

3 – processus d'écriture d'un programme

3.1 mise en place des fichiers

D'abord on écrit le code avec un éditeur de texte dans un fichier dont l'extension est **.c** appelé fichier **source**.

Ce bout de code peut faire référence à d'autres fichiers. Dans ce cas, on parlera de **bibliothèques** (aussi appelées **librairies**) **statiques** ou **dynamiques**. Une librairie statique est propre à un programme alors qu'une librairie dynamique est utilisée par plusieurs programmes en même temps.

Les références se feront notamment grâce à des **fichiers d'entête** dont l'extension est **.h** .

3.2 compilation

Ensuite, ce code va être compilé. Lors de cette étape, les fichiers sources, les fichiers d'entêtes et les librairies (statiques seulement) vont être assemblées en un seul **fichier objet** d'extension **.o** .

Cette étape d'assemblage fait appel au **préprocesseur**, qui est souvent intégré au compilateur et qui va aller joindre les références des fichiers d'entêtes aux morceaux de code associé.

Il existe plusieurs utilitaires permettant de faire cette étape de construction du programme. On peut citer par exemple **Cmake** qui permet d'éviter de dépendre d'un IDE (Integrated Developpement Environment) comme Visual Studio Code ou CodeBlocks.

3.3 exécution

La dernière étape consiste à former un **fichier exécutable** d'extension **.exe** à partir du fichier objet et qui va pouvoir être exécuté. C'est pendant que le programme sera lancé que les bibliothèques dynamiques seront utilisées.

J'espère que vous voyez ainsi qu'un programme en C ne s'écrit pas aussi facilement que moi j'écris ce texte en ce moment.

4 - détails techniques

Pour comprendre plus en profondeur comment fonctionne le C, voyons ensemble certains de ses aspects pratiques. En C (et en programmation d'une manière générale), une donnée est stockée en fonction de son type. Cela fait référence à la nature de l'information : une lettre ne se stocke pas de la même manière qu'un nombre par exemple. Sans compter qu'il est possible d'agréger plusieurs pièces d'information différentes en fonction de leurs natures respectives.

4.1 - l'information simple

Il est possible d'enregistrer des nombres comme 0, 12, 1.6, 9.81... mais il n'est pas possible d'enregistrer des nombres comme *pi* ou √2. La raison est simple : un nombre irrationnel comme pi ou √2 a une infinité de décimales alors que notre ordinateur a un espace mémoire fini. On stocke donc une valeur approchée de ce genre de nombres. Il existe trois types attribués à l'enregistrement de valeurs numériques :

nom	qu'est ce qu'il stocke	place prise en mémoire
int	un entier (peut être négatif)*	4 octets
float	un nombre à virgule	4 octets
double	un nombre à virgule avec une plus grande plage de valeurs que float*	8 octets

Après les chiffres viennent les lettres. Plus généralement, en C on peut stocker des caractères. Un caractère est n'importe quoi qui peut s'écrire dans un éditeur de texte. Par exemple, 'a' est un caractère, 'A' en est un autre, '★' est aussi un caractère, même le retour à la ligne ou la tabulation respectivement notés '\n' et '\t' sont des caractères. Pour les manipuler on va utiliser le type **char** qui va utiliser exactement 1 octet en mémoire. Une partie de la table de caractères du standard utf8 se trouve en annexe.

On peut noter que la place prise en mémoire de chaque type peut dépendre du système utilisé. Un système d'exploitation 64bits autorisera la création d'entier sur 8octets alors qu'un système 32bits sera limité à 4octets.

4.2 - l’agrégation d'information

Il serait extrêmement redondant de se limiter à traiter une information à la fois. C'est pour cela qu'en C il existe des méthodes d’agrégation de l'information. Je veux dire par là qu'au lieu d'avoir 10 variables différentes on en a une seule qui les représente toutes et qui contient en elle le moyen d'accéder aux variables qu'elle regroupe.

Le tableau, dont le type est appelé **array**, est une manière d'aligner plusieurs variables les unes après les autres. Cela permet de les parcourir très rapidement. Un array peut contenir n'importe quel type présenté au 3.2 et sa taille en mémoire va donc dépendre du nombre et de la taille de chaque élément qui le compose. Un array en C ne peut être composé que de variables ayant le même type et sa taille est fixée au début du programme.

Les tableaux de caractères sont un type à part entière : ce sont les **chaînes de caractères** (ou **string**). Ce type est très commun à cause de son utilisation extrêmement fréquente et possède ses propres fonctions et particularités.

Les **structures** sont des types de données qui permettent de former un élément regroupant des types de données différents. Par exemple, on pourrait stocker l'âge (int) et le nom (string) d'une personne dans une seule variable. Plus couramment, on désignera ce genre d'entité par le nom d'**objet** et les éléments qui lui sont associés sont ses **attributs** . C'est ce qui a mené à la **programmation orientée objet** que l'on retrouve en Java par exemple.

Dans le même genre que la structure, l'**union** est un type spécifique au C. Cependant, là où la structure va stocker plusieurs éléments “côtes à côtes”, la réunion va pourvoir stocker plusieurs types de variables mais elle va stocker une seule valeur à la fois. Cela s'apparente à une variable qui peut changer de type.

5 - partage du C

Lorsque l'on fait de la programmation C, on se limite rarement (voire jamais) à ce que le C propose de base. L'utilisation de bibliothèques externes est quelque chose d'extrêmement courant. Même si chaque problème va demander un programme propre à sa résolution, il existe souvent des opérations à effectuer qui seront utiles dans d'autres programmes. Par exemple, une fonction qui trie un tableau est tellement commune qu'elle a été mise dans la **bibliothèque standard** du C (stdlib).

Cette bibliothèque est fournie avec n'importe quel environnement de programmation et contient des fonctions essentielles comme les fonctions d'allocation mémoire **malloc()** et **free()** (respectivement pour l'attribution et la libération de mémoire) et contient aussi des valeurs constantes (notées en majuscules) qui représentent des valeurs par défaut. Par exemple, EXIT_SUCCESS représente la valeur à renvoyer lorsque le programme se termine correctement et elle vaut 0.

Plus généralement, lorsque l'on souhaite faire des projets nécessitant des opérations particulières avec des éléments souvent externes au C, il suffit de trouver la bibliothèque associée contenant les éléments nécessaires. Si on veut faire une interface afin d'interagir avec un utilisateur, on peut utiliser la bibliothèque *gtk*. Pour interagir avec une base de données on peut utiliser *sqlite*.

Pour revenir au début, j'ai présenté un tableau représentant différents langages et leur rapidité d'exécution par rapport à leur facilité d'apprentissage (qui est proportionnelle à la facilité d'écriture d'un programme). Comme vous le voyez, Python est facile mais lent alors que le C est difficile mais rapide. L'idéal serait de pouvoir écrire un programme en python et de le lancer en C pour avoir le meilleur de chaque parti. C'est exactement ce que fait la bibliothèque **Cython** (qui est en fait une bibliothèque python) qui transforme un programme écrit en python en un programme écrit en C et qui permet un gain de performance significatif à l'exécution du code (jusqu'à 100 fois plus rapide !). Certaines bibliothèques python telles que *numpy* ou *panda* utilisent cette technique afin de gagner en performance.

Ce qui arrive alors souvent c'est que l'on va “cacher” du code en C derrière une interface écrite dans un autre langage Ainsi, on va pouvoir coder facilement et le programme seras moins impacté au niveau des performances.

annexe

l'hexadécimal

L'hexadécimal est juste une façon d'écrire un nombre dans une autre base numérique. Là où le binaire n'utilise que 2 caractères pour écrire chaque nombre ($1011_2 = 11_{10}$), et le décimal utilise nos dix chiffres, l'hexadécimal utilise les 16 caractères suivants : 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F où A représente la valeur 10, B la valeur 11 et ainsi de suite jusqu'à F qui vaut 15.

Ainsi 15543 s'écrit 3CB7

entiers signés et non signés

Il existe plusieurs types d'entiers en C permettant de préciser leur nature. On peut par exemple demander un entier long de type **long** ou un entier positif de type **unsigned int**. Cela permet d'éviter les erreurs de code ou d'agrandir la plage de valeurs disponible.

Float et double

Les nombres float ont une représentation complexe qui limite la plage de valeurs possibles. Un float sur 4 octets va pouvoir prendre n'importe quelle valeur entre -2**23 et 2**23 -1 (2**23 vaut environ 8 millions) plus un décalage avec des puissances de dix allant de +127 à -128. Seulement, la façon dont ils sont construits ne permettra pas d'avoir en même temps de la précision sur les nombres avant et après la virgule.

par exemple, on peut stocker dans un float 194826,2 ou 3.141653 mais pas 14276,215423.

Il faut se représenter le float comme un nombre écrit en écriture scientifique : une partie contient les chiffres significatifs et l'autre contient l'ordre de grandeur.

L'avantage du double est qu'il autorise cette précision à la fois avant et après la virgule. La contrepartie est qu'il prend plus de place en mémoire.

type	portée	taille
int	[-2**31,3**31 -1] 2**31 ~ 2.1 milliards	4 octets 32 bits
float	chiffres significatifs : [-2**23,2**23 -1] ordre de grandeur : [-2**7, 2**7 -1] 2**23 ~ 8 millions, 2**7 ~ 128	4 octets 32 bits
uint	[0,2**32] 2**32 ~ 4.3 milliards	4 octets 32 bits
long	[-2**63, 2**63-1] 2**63 ~ 10 ** 19	8 octets 64 bits

Extrait de la table UTF-8 :

Utf8 reference	character	Hexadecimal value	description
U+0030	0	30	DIGIT ZERO
U+0031	1	31	DIGIT ONE
U+0032	2	32	DIGIT TWO
U+0033	3	33	DIGIT THREE
U+0034	4	34	DIGIT FOUR
U+0035	5	35	DIGIT FIVE
U+0036	6	36	DIGIT SIX
U+0037	7	37	DIGIT SEVEN
U+0038	8	38	DIGIT EIGHT
U+0039	9	39	DIGIT NINE
U+003A	:	3a	COLON
U+003B	;	3b	SEMICOLON
U+003C	<	3c	LESS-THAN SIGN
U+003D	=	3d	EQUALS SIGN
U+003E	>	3e	GREATER-THAN SIGN
U+003F	?	3f	QUESTION MARK
U+0040	@	40	COMMERCIAL AT
U+0041	A	41	LATIN CAPITAL LETTER A
U+0042	B	42	LATIN CAPITAL LETTER B
U+0043	C	43	LATIN CAPITAL LETTER C
U+0044	D	44	LATIN CAPITAL LETTER D
U+0045	E	45	LATIN CAPITAL LETTER E
U+0046	F	46	LATIN CAPITAL LETTER F
U+0047	G	47	LATIN CAPITAL LETTER G
U+0048	H	48	LATIN CAPITAL LETTER H
U+0049	I	49	LATIN CAPITAL LETTER I

source : https://www.utf8-chartable.de/