Bien programmer en langage C

par Emmanuel Delahaye (Site d'Emmanuel Delahaye)

Date de publication : 28 mai 2008

Dernière mise à jour :

Le langage C, bien qu'ancien, est toujours d'actualité, notamment dans le domaine du logiciel embarqué. Il est réputé pour sa puissance et son efficacité. Il dispose d'une syntaxe qui allie une certaine proximité avec la machine (mais en restant portable) et un bon niveau d'abstraction.
Son apparente simplicité cache certaines difficultés d'utilisation qu'il est bon de mettre en évidence afin d'éviter des erreurs de réalisation.
Petit florilège : **L'Echelle de Goret**.
Le but de ce site est de fournir au programmeur les moyens de réaliser du code correct, fiable et maintenable. Il s'adresse autant aux débutants qu'aux programmeurs C avertis.
En plus de l'étude de certains phénomènes dangereux du langage, et des moyens de les éviter, ce site fourni des conseils sur l'organisation du développement d'un projet en C, ainsi qu'une bibliothèque de fonctions écrites en C standard (ISO/IEC 9899:1990 ou C90) permettant de réaliser des opérations courantes de façon fiable et simple.
Ressources
Internet
Pour compléter les éléments apportés par ce site, je ne saurais trop recommander la consultation des deux FAQ C dont celle de Steve Summit en **anglais** et celle de Guillaume Rumeau en **français**. Ce sont des mines d'informations qu'il convient de lire et de relire régulièrement.
Je recommande aussi la fréquentation des deux forums USENET qui sont à l'origine de ces FAQ, soit **news:comp.lang.c** et **news:fr.comp.lang.c** ne serait-ce qu'en lecture simple. Il est bien sûr conseillé d'y poser des questions si celles-ci n'ont pas trouvé de réponses dans leurs FAQ respectives. Il existe aussi un **CLC-Wiki** qui centralise petit à petit les informations pertinentes de c.l.c.
Je signale aussi l'existence du forum privé **Développez** qui contient aussi beaucoup d'informations de bases sur le C et les autres langages de programmation.
Enfin, vous trouverez une référence détaillée et très lisible (en anglais) des fonctions de la bibliothèque standard **C90** ou **C99**
J'ajoute un document de référence non officiel mais très détaillé qui explique la norme (ISO) avec précision (en anglais) : **Le rationale**.

Spécial étudiant fauché. Voici un bon livre de C gratuit en anglais **The C Book** avec exercices et corrigés, sans oublier le **tutoriel en français** de Bernard Cassagne, et le **cours avancé** d'Anne Canteaut.
Enfin, pour les spécialistes à la recherche de structures de données plus ou moins complexes, je signale l'existence de la **Kazlib**. (en anglais).
Autre référence de haut niveau, le **site** de Chris Torek (un éminent intervenant de comp.lang.c. Toutes ses réponses sont bonnes à lire).
Je signale aussi l'existence du **Site du Zéro** qui, malgré sa présentation jeune et ludique, bénéficie d'un contenu rédactionnel correct[1]. A recommander pour les moins de 18 ans... (et même les autres, après tout). Les cours sont prolongés par un **forum C** tout à fait convenable.
Littérature
La référence incontournable : Le livre de Brian Kernighan et Dennis Ritchie, les fondateurs : Le **langage C** à compléter de l' **errata**
Autre livre intéressant, qui traite, entre autre, de C99 : **Méthodologie de la programmation en C** écrit par Achille Braquelaire.
Les apprentis gourous trouveront dans ce livre **C-Unleashed**. (en anglais) des techniques de développement C très pointues et très portables. Ecrit par de nombreux intervenants de haut niveau de comp.lang.c. "Le livre de la communauté C".
[1] bien que je ne sois pas d'accord avec certains choix de l'auteur, comme l'usage de scanf() ou des long.

I - Initiation pragmatique au langage C

I-A - Introduction

Il existe de nombreuses façons d'expliquer les bases d'un langage de programmation. J'ai choisi une approche progressive, mais concrète et basée sur l'expérimentation. Les informations données ne prétendent pas être exhaustives, et font référence à la définition du langage C selon la norme ISO/IEC 9899:1990 (appelée aussi C90). Pour les détails, il est conseillé de se munir d'un livre de référence comme ***The C programming language*** de Brian Kernighan et Dennis Ritchie (ISBN 2-100-05116-4) et son **correctif** ou sa version française : **Le langage C**. Des solutions aux exercices sont dévoilées **ici**.

Le langage C est un langage normalisé qui comprend des instructions de **préprocesseur**, des instructions C, et des symboles dont la sémantique peut varier selon le contexte.

I-B - Programme minimum

Voici le programme C le plus simple que l'on puisse écrire.

```
int main (void)
{
   return 0;
}
```

Ce programme est composé de plusieurs mots clés faisant partie du langage C : **int**, **void** et **return**

Il utilise aussi 5 symboles syntaxiques, à savoir **(,)**, **{, }** et **;**. Enfin, il utilise un mot défini par l'utilisateur : main

Ce programme ne fait rien de visible. Cependant, il met en oeuvre la structure de code de base du langage C, à savoir la **fonction**.

I-B-1 - Analyse détaillée

Nous allons étudier les mots et le symboles qui constituent le programme dans l'ordre de leur apparition.

int est un **type**. Ici, il qualifie la valeur retournée par la fonction.

main est un identificateur défini par l'utilisateur. Utilisé dans ce contexte, il désigne le nom de la fonction. Cet identificateur a cependant un sens particulier. Il designe le point d'entrée du programme. Cela signifie qu'un programme C commencera toujours par un appel 'invisible' à la fonction main(). Cet appel provient de la séquence d'initialisation du programme. Les détails dépendent de l'implémentation, mais, d'une manière simplifiée, voici quel est le déroulement des opérations:

1 chargement du logiciel
2 execution du code d'initialisation ("boot")
3 appel de l'application (CALL MAIN)
4 execution de l'application, jusqu à la fin de main()
5 exécution du code de fin
6 retour au système.

void est un mot clé qui signifie 'rien' ou 'absence de'. Ici, il sert à indiquer que la fonction n'a pas de **paramètres**.

return est un mot clé qui signifie "quitter la fonction courante". Si la fonction n'a pas de type de retour (type **void**) on peut utiliser **return** tout seul suivit d'un point-virgule. Sinon, il doit être suivi d'une valeur et d'un point-virgule. La valeur doit être du type qui a été défini pour la fonction.

Cette valeur peut avoir un sens particulier. Il est d'usage de retourner 0 pour signifier que tout va bien (pas d'erreur) et une valeur autre que 0 pour signifier une erreur.

*L'usage d'un 'return' sans valeur de retour dans une fonction qui attend une valeur de retour provoque un **comportement indéfini**.*

I-C - Un programme qui dit "bonjour"

Voici une version un peu plus élaborée, qui se contente d'afficher une simple phrase ("Hello world!") sur la sortie standard (généralement l'écran).

```
#include <stdio.h>

int main (void)
{
   puts ("Hello world!");
   return 0;
}
```

Dans ce programme, on constate l'ajout du mot clé **#include** et de son paramètre <stdio.h>, ainsi que la fonction puts(). et de son paramètre "Hello world!".

On remarque que le mot clé **#include** commence par un '#'. En effet, ce mot clé appartient à la famille des commandes du préprocésseur, dont une caractéristique est justement de commencer par '#'.

#include<stdio.h> signifie que le fichier indiqué en paramètre est inclus dans le code source. Cette opération (ainsi que toutes les opérations du préprocesseur) est effectuée avant la traduction du code.

Cette opération d'inclusion est rendue nécessaire à cause de l'appel à la fonction puts(). En effet, il est recommandé (et parfois obligatoire) de fournir au compilateur un **prototype** à la fonction utilisée :

```
int puts (char const *);
```

La fonction puts() appartient à la bibliothèque standard du langage C, qui fait partie intégrante du langage. La définition du prototype de cette fonction se trouvant dans le fichier d'en-tête **<stdio.h>**, il est donc tout à fait logique d'inclure ce fichier dans le fichier source.

Cette fonction a pour effet d'émettre la chaîne de caractères passée en paramètre vers le **flux** de sortie standard. Elle ajoute automatiquement un caractère de fin de ligne.

A première vue, le paramètre de cette fonction est **une chaîne de caractères**, En fait, c'est l'adresse de celle-ci. C'est pourquoi le paramètre de fputs() est défini comme un pointeur sur un type **char**. En effet, un pointeur est une variable qui peut contenir une adresse.

Le qualificateur **const** signifie que la fonction accepte l'adresse d'une chaîne non modifiable, car elle s'engage à ne pas modifier la chaîne pointée, ni à passer son adresse à une fonction qui pourrait la modifier.

I-D - Glossaire

I-D-1 - Bit

Le bit (BInary digiT) ou chiffre binaire est l'unité de mesure d'information. Il peut prendre 2 valeurs : 0 ou 1.

I-D-2 - Byte

Le byte (ou multiplet) est le plus petit objet adressable pour une implémentation donnée. Il fait au moins huit bits.

I-D-3 - Caractère

Un caractère est une valeur numérique qui représente un glyphe visible ('A', '5', '*' etc.) ou non (CR, LF etc.). Un caractère est de type **int**. Cependant, sa valeur tient obligatoirement dans un type char.

Un objet de la taille d'un byte peut recevoir la valeur de n'importe quel caractère.

I-D-4 - Chaîne de caractères

Une chaîne de caractères est une séquence de **caractères** encadrée de double quotes.

```
"Hello world!"
```

La représentation interne d'une chaîne de caractères est spécifiée. C'est un **tableau** de caractèes (char) terminé par un 0.

```
char s[] = "Hello";
```

est équivalent à :

```
char s[] = {'H', 'e', 'l', 'l', 'o', 0};
```

I-D-5 - Comportement indéfini

Un comportement indéfini (encore appelé *Undefined Behaviour* ou *UB*) est un bug grave. Il signifie que le comportement du programme est imprévisible et que tout peut arriver, y compris, et c'est ça qui rend ce bug très dangereux, un comportement d'apparence conforme.

Il n'existe pas de moyen automatique de détecter tous les UB d'un programme. Seul un contrôle visuel fait par une personne chevronnée permet de détecter un tel bug.

I-D-6 - Fonction

Une fonction est une séquence d'instructions dans un bloc nommé. Une fonction est constituée de la séquence suivante :

- Un type (ou le mot clé void).
- Un identificateur (le nom de la fonction, ici main).

- Une parenthèse ouvrante **(.**
- Une liste de paramètres ou une liste vide (dans ce cas, on écrit void).
- Une parenthèse fermante **)**.
- Une accolade ouvrante **{**.
- Une liste d'instructions terminée par un point virgule **;**, ou rien.
- Une accolade fermante **}**.

L'utilisateur peut créer ses propres fonctions. Une fonction peut appeler une autre fonction. Généralement, une fonction réalisera une opération bien précise. L'organisation hiérarchique des fonctions permet un raffinement en partant du niveau le plus élévé (main()) en en allant au niveau le plus élémentaire (atomique).

On gardera cependant en tête que la multiplication des niveaux introduit une augmentation de la taille du code et du temps de traitement.

Une fonction ne peut être appelée qu'à partir d'une autre fonction.

I-D-7 - Flux

Un flux est un canal de données orienté **byte**. Il permet de réaliser des entrées ou des sorties de bytes, soit un par un, soit par bloc.

C'est le mécanisme du langage C qui permet l'interaction avec l'environnement.

Un programme C ouvre 3 flux par défaut :

- stdin Le flux d'entrée standard
- stdout Le flux de sortie standard
- stderr Le flux de sortie d'erreur

Sur la plupart des implémentations, ces flux sont connectés à la console.

I-D-8 - Paramètre

La zone entre parenthèses d'une fonction peut recevoir des paramètres. Un paramètre est défini au minimum par un type et un identificateur. Exemple :

```
int fonction (int x)
{
}
```

x est un paramètre de type **int**. Il est obligatoire, et la valeur passée doit être du même type.

```
...
{
   fonction (123);
}



...
{
   int a = 123;
```

```
   fonction (a);
}
```

Il faut savoir qu'en C, un paramètre ne fait que transmettre une valeur. Modifier la valeur d'un paramètre n'aura jamais d'effet sur la valeur originale. (la fonction **printf()** permet ici de visualiser la contenu des variables a et x) :

```
#include <stdio.h>

void fonction (int x)
{
   printf ("x = %d (dans la fonction)\n", x); /* x = 123 */

   x = 456;
   printf ("x = %d (dans la fonction)\n", x); /* x = 456 */
}

int main (void)
{
   int a = 123;

   printf ("a = %d\n", a); /* a = 123 */

   fonction (a);
   printf ("a = %d\n", a); /* a = 123 */
   return 0;
}
```

Après l'appel de la fonction, la valeur de 'a' est inchangée (123).

```
a = 123
x = 123 (dans la fonction)
x = 456 (dans la fonction)
a = 123
```

Le langage C permet la modification, mais en utilisant des techniques plus avancées.

I-D-9 - Préprocesseur

Le préprocesseur est un outil qui traduit certaines instructions du code source avant le compilateur. Les instructions du préprocesseur commencent par #. Par exemple **#include**

I-D-10 - printf()

printf() est une fonction standard de la bibliothèque du la biliothèque du C. Elle est conçue pour émettre des chaînes de caractères formatées vers la sortie standard (*stdout*). Elle est déclarée dans le fichier d'entête <stdio.h>. Pour l'utiliser, on doit utiliser la directive **#include <stdio.h>**.

Cette puissante fonction permet de réaliser des affichages formatés. Sa descrition complète prendrait un chapitre entier. Voici donc quelques éléments de base.

Elle fonctionne selon un principe un peu particulier :

Son premier paramètre est une chaîne de caractères pouvant comporter des séquences spéciales commençant par "%". Le caractère qui suit le "%" a une signification particulière. Certaines sont définies (%, d, i, f, s, c etc.), d'autres

non. Le rôle est cette séquence de caractères est d'indiquer à printf() comment il doit interpréter les valeurs passées dans les paramètres suivants. Il doit y avoir correspondance, sinon, le comportement est indéfini.

Formateur de base	Rôle	type attendu	Types compatibles
%	affiche le glyphe du caractère %	int	short, char
c	affiche le glyphe d'un caractère imprimable	int	short, char
d	affiche la valeur d'un entier en décimal	int	short, char
f	affiche la valeur d'un reel en décimal avec virgule fixe	double	float
s	affiche les glyphes d'une chaîne de caractères	char *	

Exemples d'utilisation de printf()

```
#include <stdio.h>

int main (void)
{
   printf ("%c\n", 'A');
   printf ("%d\n", 123);
   printf ("%d, '%c'\n", 'A', 'A');

   printf ("Hello world\n");
   printf ("Hello %s\n", "world");
   printf ("%s\n", Hello world);

   return 0;
}
```

produit :

```
A
123
65, 'A'
Hello world
Hello world
Hello world
```

Les possibilités complètes de cette puissante fonction sont décrites dans un livre de C comme ceux recommandés au début de ce chapitre.

I-D-11 - Prototype

Le prototype d'une fonction est une déclaration indiquant:

- Le type du retour (ou void si il n'y en a pas)
- Le nom de la fonction
- La liste et le type des paramètres (ou void si il n'y en a pas).

Un prototype peut être integré à la fonction. Dans ce cas, il est confondu avec la première ligne de la fonction :

```
int function (void)
```

```
{
   /* ... */
   return 0;
}
```

Il peut aussi être détaché (séparé) de la fonction. Dans ce cas, il doit être suivi d'un ';'.

```
int function (void);
```

I-D-12 - sizeof

L'opérateur unaire **sizeof** retourne la taille d'un objet en bytes. Le paramètre de **sizeof** peut être un objet ou un type. Si c'est un type, celui-ci doit être placé entre parenthèses. Le type retourné par sizeof est size_t (entier non signé). L'expression est une constante évaluée à la compilation.

size_t est un type standard défini dans <stddef.h>.

```
#include <stddef.h>

size_t size_of_an_int = sizeof (int);

double n;

size_t size_of_n = sizeof n;
```

I-D-13 - Tableau

Un tableau est une suite d'objets identiques consécutifs.

```
/* Definition d'un tableau de 10 char */
char tab[10];
```

Il est possible d'initialiser un tableau au moment de sa définition. Les membres non initialisés seront forcés à 0 :

```
/* Definition d'un tableau de 20 entiers
 * de type long avec initialisation a 0
 */
long tab[20] = {0};

/* Definition d'un tableau de 5 flottants
 * de type double avec initialisation partielle
 */
double tab[5] = {12.34, 56.78};
```

Enfin, il est possible de déterminer la taille d'un tableau par son initialisation. La taille s'ajuste alors en fonction du nombre d'initialiseurs :

```
/* Definition d'un tableau de d'entiers
 * de type int initialises
 */
int tab[] = {1, 2, 3, 4, 5, 6};
```

Gràce à l'opérateur **sizeof**, il est possible de déterminer le nombre d'élements d'un tableau. Il suffit de diviser la taille du tableau (en bytes) par la taille d'un élément du tableau (en bytes) (par exemple, [0], mais la valeur exacte de l'indice est sans importance).

```
/* Nombre d'elements de 'tab' */
size_t nb_elem = sizeof tab / sizeof tab[0];
```

I-D-14 - Type

Le type est une des propriétés qui caractérise une valeur. Elle permet de préciser quelle est la gamme de valeurs possible. La norme qui définit le langage C précise qu'en fonction du type, ces valeurs doivent couvrir une gamme minimale.

Il y a quatre types en langage C:

- char : petit entier, convient pour les caractères
- int : pour les valeurs numériques entières
- float : nombres à virgule flottante en simple précision
- double : nombres à virgule flottante en double précision

Voici les types entiers courants avec leur gammes de valeurs minimales. Les termes entre [] sont facultatifs.

Type	Minimum	Maximum
char	0	127
unsigned char	0	255
signed char	-127	127
[signed] short [int]	-32767	32767
unsigned short [int]	0	65535
[signed] int	-32767	32767
unsigned [int]	0	65535
[signed] long [int]	-2147483647	2147483647
unsigned long [int]	0	4294967295
[C99] [signed] long long [int]	-9223372036854775807	9223372036854775807
[C99] unsigned long long [int]	0	18446744073709551615