Enseignant
Manuel clergue

1

Introduction au langage C
Les bases

2022

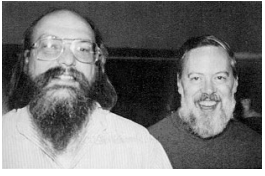
esiea

1 Le langage C

Le langage C ...

- est un langage impératif vénérable (début des années 70)
- et relativement stable (ANSI C / C89 → C99 → C11 → C18 → ···)
- il est syntaxiquement simple et assez bas niveau
- avec un grand nombre de bibliothèques
- il permet d'écrire des programmes très rapides et assez compacts (peu de vérification)

Cela explique son succés et qu'il soit encore très utilisé ...
(`https://www.tiobe.com/tiobe-index/`)



Ken Thompson et Dennis Ritchie en 1973

Un programme minimal

```
1  #include <stdio.h>
2
3  int main(){
4    printf ("Hello World\n");
5    return 0;
6  }
```

Instructions préprocesseur

Point d'entrée du programme

Les bonnes pratiques

- Réfléchir avant de coder !
  - Comprendre et analyser le problème
  - Ne pas hésiter à faire des schémas sur papier
- S'assurer que le code est lisible !
  - Respecter une convention de nommage/style
  - Utiliser des noms de variables et de fonctions explicites
  - Indenter
  - Mettre des commentaires (`/* ... */` ou `// ...`)
  - Ne mettre qu'une seule instruction par ligne
  - Eviter les valeurs en dur
- Tester le code !
  - Tests unitaires
  - *Review de code*
  - *Outils de debug, de profilage, de détection des fuites mémoires*

Les bonnes pratiques

- Eviter les redondances :
  - Valeurs : constantes / fichiers de données
  - Code : macros / fonctions / modules
- Il existe trois niveaux d'erreurs
  - Erreur de compilation (facile à repérer, assez simple à traiter)
  - Erreur d'exécution (un peu moins facile à repérer, beaucoup moins simple à traiter)
  - Le programme ne répond pas aux spécifications (difficiles à repérer)
- Protéger vos entrées et vos appels de fonctions
  - Lecture clavier et fichier : vérifier que les données acquises sont conformes
  - Retour de fonctions *systèmes* (ouverture de fichier, réservation mémoire, ...)

N'importe qui peut écrire du code qu'un ordinateur peut comprendre. Un bon programmeur écrit du code qu'un humain peut comprendre.

Toujours écrire votre code comme si la personne qui va le maintenir avait des tendances meurtrières, et qu'il sait où vous habitez.

2 La syntaxe du C

Les identifiants

- Un identifiant est le nom qu'on donne à un objet pour pouvoir le référencer plus tard :
  - Variable
  - Fonction
  - Macros
  - User-defined types
  - ...
- Il est composé de au plus 31 caractères :
  - Des lettres sans accent (case sensitive)
  - Des chiffres (interdit au début)
  - Des caractères `_` (à éviter au début)
- Il ne doit pas être un mot réservé du langage

Les mots réservés du langage C

les spécificateurs de type	char double float int long short signed unsigned void
les classes d'allocation	auto register static extern
les constructeurs	enum struct typedef union
les qualificateurs de type	const volatile
les instructions de contrôle	break case continue default do else for goto if switch while return
divers	inline restrict sizeof

Les valeurs (constantes littérales)

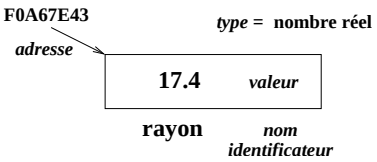
- Les nombres entiers
  - entiers : `4` , `0` , `-4`
  - avec une qualification en suffixe : `4U` , `-10L` , `50UL`
  - bases non décimales : `010` (octal), `0xFF` (hexadécimal)
- Les nombres réels
  - simple précision : `5.2f` `-5.f`
  - double précision : `5.2` `5.` `0.` `0.0`
  - en notation *scinetifique* : `-5.2E-2`
- Les caractères
  - normaux : `'a'`
  - par code ASCII : `'\10'` (octal), '\x1A' (hexadécimal)
  - spéciaux : '\n' '\t' '\r' '\'' '\"' '\\'
- Les chaïnes de caractères
  - `"La chaîne  \"Laval\" est un palindrome\n"`
  - Attention sur les chaînes de caractères (on y reviendra . . .)

Les constantes nommées (préprocesseur)

- Il est possible de définir des constantes nommées
- Elles seront référencées par leur nom
- Elles ne peuvent pas changer de valeur !
- Instruction préprocesseur : `#define MACONSTANTE 10`
- c'est juste un search-and-replace : le préprocesseur remplace la chaîne (`MACONSTANTE`) par la valeur (`10`)
- Il est préférable d'avoir des noms de constante en majuscule

Les **variables** servent à représenter les données, les résultats, etc...

Une **variable** correspond à :



- (pour l'homme) un objet, une quantité
  - qui reçoit un **nom** ou **identificateur**,
  - qui est d'un certain **type** (selon la nature de l'information),
  - qui peut prendre une **valeur** (susceptible de changer) ;
- (pour la machine) un emplacement mémoire
  - caractérisé par une **adresse** (lieu où la variable est stockée)
  - et une taille (place utilisée) qui dépend du type de la variable.

Les variables

- La valeur peut varier au cours de l'éxécution du programme
- Il faut distinguer deux opérations :
  - La déclaration : création de la variable par association d'un nom et d'un type `int a;`
  - L'affectation : la modification de la valeur de la variable `a=10;`
  - (il est possible de grouper les deux : `int a=10;`)
- Dans une opération impliquant une variable, à l'exécution, la variable est remplacée (évalué) par sa valeur : `a+2` $\rightarrow 10 + 2$
- Il n'est pas possible de rédéclarer une variable (dans son *scope*)
- Les variables n'ont pas de valeur par défaut

mot-clé	type	bornes
int	Entier	-32768 à 32767, si processeur 16 bits
		-2147483648 à 2147483647, si 32 bits
float	flottant (Réel)	$3,4 \times 10^{-38}$ à $3,4 \times 10^{38}$
double	flottant double	$1,7 \times 10^{-308}$ à $1,7 \times 10^{308}$
long double	double long	$3,4 \times 10^{-4932}$ à $3,4 \times 10^{4932}$
char	Caractère (1 seul)	-128 à 127 (codage ASCII)

Il existe aussi des long int, short int, unsigned int, unsigned char, etc ...
voir par exemple :

`https://fr.wikibooks.org/wiki/Programmation_C/Types_de_base`

WARNING :

sur un ordinateur, quels que soient la précision du processeur, le langage utilisé, etc. . .il existe des limitations sur les nombres que la machine peut coder :

- il existe un plus grand entier positif et un plus petit entier négatif (en général son opposé, à 1 près),
- idem pour les nombres réels,
- il existe un plus petit réel strictement positif, i.e. il y a un intervalle entre 0 et ce nombre $\Rightarrow$ risques d'erreurs d'arrondi.

voir :
`https://fr.wikibooks.org/wiki/Fonctionnement_d%27un_ordinateur/Codage_des_nombres`
pour plus d'informations sur le codage des nombres.

Manipulation de variables : un exemple

```
1  // constante
2  #define PI 3.14
3
4  // Déclarations
5  int a;
6  int b=10;
7  int i, j=0, k=7;
8  unsigned char c;
9  float pi_val=PI, f=0.3;
10
11 // Affectations
12 i = 12;
13 c = 'c';
14 i = j + 1;
15 i += 7; // Peut aussi s'écrire i = i + 7
16
```

Transtypage(cast, forcage de type)

- Il est possible de convertir une variable vers un autre type.
- Deux possibilités :
  - Forçage implicite (éviter)
    - Conversion vers le type de meilleure précision lors d'une évaluation.
    - Conversion vers le type de l'opérande de gauche lors d'une affectation.
  - Forçage explicite : Le type voulu est explicité entre parenthèses avant la valeur/variable à convertir.

```
1  // déclarations
2  int i=7;
3  float f=0.3;
4
5  f = i + 0.6;     // implicite : Résultat flottant
6  f = f + i;       // implicite : Résultat flottant (génère un warning)
7  i = i + f;       // implicite : Résultat entier (génère un warning)
8  i = (int)f + i; // explicite : Résultat entier
```

Constantes (variables)

Il est possible de définir des variables constantes (!)
Ce sont des variables dont la valeur ne change pas au cours de l'exécution du programme
Elles se définissent par le mot clé `const` :

```
1  const int une_constante = 10;
2
3  une_constante = 11; //  error: assignment of read-only variable ‘une_constante’
```

3 Entrées/Sorties (part I)

Programmer un affichage à l'écran :

printf("chaîne-format",liste d'arguments) ;

La chaîne-format est constituée de texte, de caractères de contrôle et de spécifications de formats pour l'affichage des variables.
La liste d'arguments, éventuellement vide, contient les valeurs qui doivent être affichées, en respectant le nombre, l'ordre et les types des spécifications contenues dans la chaîne-format.

Exemple :

```
printf("L'aire mesure %f centimétres carrés.\n" ,aire);
```

affiche du texte et se termine par un passage à la ligne ; la valeur de la variable **aire** (qui doit être de type float) sera écrite au milieu du texte.

Attention :

Les fonctions d'entrée/sortie ne sont pas dans le langage de base. Il faut inclure une bibliothèque :

```
#include <stdio.h>
```

Caractères de contrôle

Déplacement de la position d'écriture :

caractère	effet produit
\n	passe au début de la ligne suivante
\r	retourne au début de la ligne courante
\t	tabulation (remplie par des espaces)
\b	retourne à la position précédente

Ecriture de caractères réservés :

ce qu'il faut taper	caractère obtenu
\"	pour écrire des guillemets dans le texte
\\	pour écrire un antislash (backslash)
%%	pour écrire le symbole % dans le texte

Spécifications de formats

Pour annoncer le type de la variable :

spécification	type de la variable
%d	entier décimal (en base 10)
%x	entier hexadécimal (en base 16)
%f	flottant (réel)
%lf	flottant double
%c	caractère (1 seul)
%s	chaîne de caractères

Pour préciser la place à réserver :

On peut insérer un nombre entier entre le % et la lettre pour dire quel nombre minimum de positions (places d'écriture) il faudra réserver pour l'affichage de la variable.

Consulter `https://koor.fr/C/cstdio/fprintf.wp` pour avoir les différents formats et leurs option !

Lecture d'une donnée qui sera tapée au clavier :

scanf("spécification du format",&nom_de_variable) ;

Exemple :

```
scanf("%d",&n);
```

attend que l'utilisateur tape un entier au clavier et place sa valeur dans la variable n (qui doit être de type int).

N.B. La présence du & devant le nom de la variable est obligatoire car il fait référence à l'emplacement mémoire à laquelle se trouve cette variable.

La fonction scanf est à éviter pour la lecture d'une chaîne de caractères !