Chapitre 1: Généralités du langage C

Babacar DIOP

Adresse Mail: diopbabacar4888@gmail.com

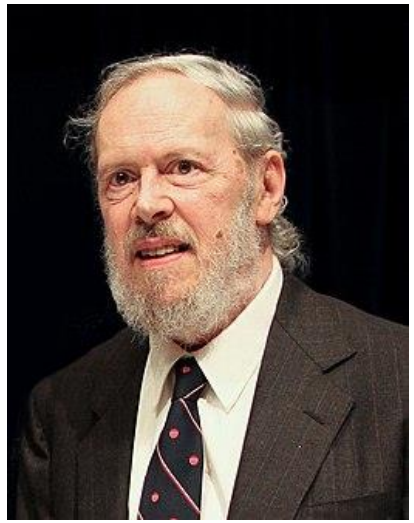
Téléphone: 777812040

Historique de la programmation

- Au tout début de l'informatique, les ordinateurs étaient des machines énormes et coûteuses utilisées principalement par des scientifiques et des militaires.
- La **programmation** était faite en langage machine, une série de codes binaires compréhensibles uniquement par les ordinateurs.
- C'était **très compliqué et propice aux erreurs**: Imaginez que vous deviez donner des instructions à un robot en utilisant uniquement des combinaisons de 0 et de 1.
- Pour simplifier la programmation, des **langages de bas niveau** comme l'assembleur, le langage C ont été créés. Ces langages étaient plus compréhensibles par les humains, mais ils étaient toujours assez proches du langage machine. Utiliser l'assembleur serait par exemple comme donner des instructions au robot en utilisant un langage un peu plus compréhensible, mais toujours assez technique.
- Les programmeurs voulaient rendre la programmation encore plus accessible. C'est ainsi que des **langages de haut niveau**, comme Python, Java, et JavaScript ont été développés. Ces langages permettaient aux programmeurs de s'exprimer avec des mots et des symboles plus proches du langage humain. Pensez à utiliser une sorte de langage de robot plus avancé, où vous pourriez dire au robot de "prendre la tasse et la placer sur la table" au lieu de donner des instructions plus complexes.
- Les sites web, applications et logiciels que nous utilisons tous les jours sont le résultat de la programmation.
- Chaque application sur votre téléphone, chaque site web que vous visitez, est le produit de la programmation. C'est comme si chaque fonctionnalité de votre robot était créée par des instructions précises que quelqu'un a écrites.

Qu’est-ce que le langage C ?

- Le langage C a été inventé par **Dennis Ritchie** au laboratoire Bell en **1972**. Son développement a été motivé par la nécessité de créer un langage de programmation plus portable et flexible que le langage assembleur, permettant ainsi de simplifier le développement du système d'exploitation UNIX. Ritchie a travaillé en collaboration avec Ken Thompson, et leur création a rapidement gagné en popularité grâce à sa simplicité et à son efficacité dans le développement logiciel.



- Le langage C occupe une place fondamentale dans l'informatique en tant que **langage système**. De nombreux systèmes d'exploitation, compilateurs et utilitaires sont écrits en langage C en raison de sa proximité avec le matériel et de son efficacité. Il a également influencé de nombreux langages ultérieurs, contribuant à la création de familles de langages comme C++ et C#.

- Apprendre le langage C est bénéfique aux débutants dans la programmation avec sa syntaxe simple et concise facilitant l'apprentissage des concepts de base de la programmation, tout en encourageant une pensée logique et structurée.

- La maîtrise du langage C fournit une base solide pour explorer d'autres langages et domaines de développement logiciel, offrant ainsi une perspective approfondie de la **programmation informatique**.

- Un **programme informatique**, décrit un ensemble d’instructions (opérations) destinées à être exécutées par un ordinateur pour régler un problème bien défini.

- L’écriture de cette démarche dans un langage de programmation constitue la brique élémentaire de la programmation informatique.

- **Dans ce module, nous allons apprendre petit à petit les concepts de base de la programmation en langage C.**

Quelles sont les caractéristiques du langage C ?

- Le langage C est écrit dans un fichier portant l'**extension .c**

  Ce fichier, appelé **fichier source du programme**, contient généralement les parties suivantes :
  - directives de compilation
  - définition des structures et des types (facultatif)
  - déclaration des variables globales (facultatif)
  - définition des sous programmes ou fonctions (facultatif)
  - programme principal (fonction main)

- Le langage C est un **langage compilé:** traduit le code source en langage machine via un compilateur avant l'exécution pour améliorer l'efficacité

- Le langage C est un **langage typé:** exige que chaque variable soit déclarée avec un type spécifique (comme int, float, char) pour une gestion rigoureuse des données et la prévention des erreurs de type.

- Le langage C est un **langage déclaratif:** tout objet manipulé par le programme doit être déclaré avant son utilisation

- Le langage C propose un certain nombre de **fonctions prédéfinies** dans des bibliothèques.

I. Structure d'un programme C
1. Programme principal

- C'est la fonction qui démarre le programme: le **programme principal**.

- En C, toutes les fonctions ont la même structure :
  - le type de retour
  - le nom de la fonction
  - la liste d'arguments mis entre parenthèses et séparés par des virgules
  - le corps de la fonction contenu entre accolades.

- Il doit y avoir une fonction main dans tout programme C.

- Le programme principal suit cette structure, mais son type de retour, son nom et ses arguments sont imposés

```c
#include <stdio.h>

int main([void]){
    /* déclaration des variables utilisées dans la fonction main */


    /* instructions du programme*/

    return 0;
}
```

- Le nom du programme principal est main;
- Son type de retour est entier (int).
- Il ne prend en général pas d'arguments.
- Dans ce cas, on met le mot void entre les parenthèses (ou laisser vide)
- Dans certains cas il peut en prendre des arguments.

I. Structure d'un programme C

2. Exemple de programme en C: « Hello, world !»

```c
#include <stdio.h>

int main() /* programme principal*/
{
    printf("Hello, World !");

    return 0;
}
```

- Dans ce programme, **#include** est une directive de compilation qui indique qu'on va utiliser des fonctions de la bibliothèque standard stdio.h (STandarD Input/Output). Une bibliothèque (ou librairie) standard est un fichier dans lequel est défini un ensemble de fonctions prêtes à l'emploi facilitant ainsi le travail du programmeur.

- Dans la fonction main, on appelle la fonction **printf**, qui est définie dans la bibliothèque stdio.h pour afficher à l'écran la chaîne de caractères **Hello, World !** Comme toute instruction, l'appel à la fonction printf se termine par un **point virgule virgule**.

- L'instruction suivante termine la fonction main qui retourne le résultat 0.

- On peut insérer des commentaires, soit entre les balises **/* */**, soit après **//** jusqu'à la fin de la ligne. Les commentaires sont des parties du programme qui sont ignorées par le compilateur. Ils servent à rendre un programme compréhensible.

I. Structure d'un programme C

3. Les directives de compilation

Elles indiquent au compilateur de procéder à des opérations préalables au début de la compilation. Ces directives se situent en tout début du programme source.

a. La directive #include

Cette directive permet l'inclusion de librairies dont les éléments seront utilisés dans le programme source. Le compilateur C fournit un ensemble de librairies mais le programmeur peut aussi créer ses propres librairies.

- Syntaxe de cette directive :

#include <fichier>
ou
#include "fichier"

<fichier>

Indique l'utilisation d'une librairie, fournie par le compilateur C, se trouvant dans un répertoire particulier du système.

Les librairies les plus utilisées sont généralement :
- sdtio.h qui contient les définitions des fonctions d'Entrée/Sortie (io pour Input/Output).
- string.h qui contient les définitions des fonctions traitant des chaînes de caractères.
- stdlib.h qui contient les définitions de fonctions traitant la conversion de nombres et l'allocation de mémoire.
- ctype.h qui contient les définitions de fonctions traitant la conversion de caractères.

"fichier"

Indique l'utilisation d'une librairie contenue dans "fichier". Cette librairie est souvent un ensemble de fonctions propres au programmeur ou un fichier de déclaration des variables du programme.
"fichier" peut contenir juste le nom de la librairie ou le chemin d'accès et le nom si elle ne se trouve pas dans le même répertoire que le programme source.

- Exemples : ***#include***
#include "C:\MesProgrammes\definitions.h"

I. Structure d'un programme C

3. Les directives de compilation

b. La directive #define

- Cette directive permet de remplacer dans le programme toutes les occurrences d'une suite de caractères par un nom de substitution.
- Cette possibilité permet une meilleure lisibilité du programme source, évite de devoir réécrire à chaque fois une suite longue de caractères souvent utilisés au cours du programme source et permet de définir des constantes pour le programme ou des macros.
- Syntaxe de cette directive :

#define nom_de_substitution suite_de_caractères

nom_de_substitution sera utilisé tout au long du programme source pour représenter la suite de caractères *suite_de_caractères*.

- Exemple:

#define VRAI 1

Le mot VRAI sera utilisé pour représenter le chiffre 1 dans le programme source. Mais rien ne vous empêche d'utiliser le chiffre 1.

II. Les types de base du langage C

- A une variable est associée un type, qui permet de définir la taille de l'espace occupé en mémoire (nombre d'octets).
- En C, les types de base, dit aussi types primitifs sont divisés en deux classes: les entiers et les réels (ou flottants).

1. Les entiers

Type	Description	Taille en octets	Valeurs possibles	Format
int	Entier standard	2 ou 4	de -32768 à 32767 (sur 2 octets) de -2147483648 à 2147483647 (sur 4 octets)	%d
short	Entier court	2	de -32768 à 32767	%d
long	Entier long	4	de -2147483648 à 2147483647	%ld

Attention : Ces tailles varient en fonction du système d'exploitation, du compilateur ou du processeur utilisé.

- L'opérateur **sizeof(type)** retourne le nombre d'octets utilisés pour stocker une valeur d'un type donné.
- Sur certains systèmes et compilateurs, **int** est synonyme de **short**, sur d'autres il est synonyme de **long**.
- Si l'on désire manipuler des entiers non signés, alors on ajoutera le mot « **unsigned** »
- Le type booléen n'existe pas en C. Par contre, la valeur 0 (au sens binaire de la représentation) de n'importe quel type signifie faux. Toute autre valeur est vraie.

2. Les réels

Type	Description	Taille en octets	Format
float	Réel	4	%f
double	Réel double précision	8	%lf

II. Les types de base du langage C

3. Les caractères

❖ Les caractères sont considérés comme des entiers codés en mémoire sur 1 octet. A chaque caractère est associé un nombre entier (son code ASCII). Ainsi, toutes les opérations s'appliquant aux entiers peuvent également s'appliquer aux caractères.

Type	Description	Taille en octets	Valeurs possibles	Format
char	Caractère (considéré comme un entier signé)	1	de -128 à 127	%c
unsigned char	caractère (considéré comme un entier non signé)	1	De 0 à 255	%c

En C, la notion de caractère dépasse celle de caractère imprimable, c'est à dire auquel est associé un graphisme (lettres, chiffres, ponctuation, caractères spéciaux,…). Par exemple, les retours chariot, les tabulations, les sauts de page sont aussi des caractères.

Les caractères non imprimables possèdent une représentation conventionnelle utilisant le caractère \ nommé antislash.

Exemples de caractères non imprimables :

- **\n** retour chariot avec saut de ligne
- **\r** retour chariot sans saut de ligne
- **\t** tabulation horizontale
- **\v** tabulation verticale

III. Déclaration des variables et des constantes

1. Déclaration d'une variable

- En C, toute variable doit être déclarée avant sa première utilisation.
- La syntaxe de déclaration d'une variable est **type identificateur;**

  Exemple:
  L'instruction: **int a ;**
  permet de réserver un emplacement mémoire pour le stockage d'un entier (type **int**) qui sera nommé **a** dans la suite du programme.

- Une déclaration peut se faire seule, ou accompagnée d'une initialisation (première valeur).

  Exemple :
  L'instruction **int b=10;**
  permet de déclarer un entier b et de lui donner 10 comme première valeur.

- On peut déclarer plusieurs variables de même type dans une même instruction en séparant les identificateurs par des virgules.
  Exemple :
  int x, y = 2, z ;

III. Déclaration des variables et des constantes

2. Déclaration d'une constante

- La syntaxe de déclaration d'une constante est: **const type identificateur = valeur;**

  Exemple **: const int TVA = 0.18;**

- On peut utiliser la directive de compilation **#define** pour définir une constante selon la syntaxe ci-dessous:

  #define identificateur valeur

- Mais dans ce cas, il n'y a pas de réservation d'emplacement mémoire.

  Exemple : **#define TVA O.18**

III. Déclaration des variables et des constantes

3. Contraintes sur les identificateurs

- Un identificateur est une suite de caractères parmi :
  - les lettres (minuscules ou majuscules, mais non accentuées),
  - les chiffres,
  - le caractère « souligné » (_).

  Le premier caractère d'un identificateur ne peut pas être un chiffre.

- Le langage C fait la différence entre les majuscules et les minuscules (la casse).

- Un certain nombre de mots, appelés mots-clefs, sont réservés pour le langage lui-même et ne peuvent pas être utilisés comme identificateurs.

- Le C standard (ANSI C) compte 32 mots clefs : **auto const double float int short struct unsigned break continue else for long signed switch void case default enum goto register sizeof typedef volatile char do extern if return static union while**

IV. Les opérateurs

1. Opérateurs arithmétiques

Opérateur	Signification
+	Addition
-	Soustraction
*	Multiplication
/	Division
%	Reste de la division entière (entre deux entiers)

2. Opérateurs relationnels

Opérateur	Signification
>	Supérieur
>=	Supérieur ou égal
<	Inférieur
<=	Inférieur ou égal
==	Egal
!=	Différent

IV. Les opérateurs

3. Opérateurs logiques

Opérateur	Signification
&&	Et logique
||	Ou logique
!	Négation logique

Exemples :
(a > 10) **&&** (a <= 20) sera vrai si a > 10 et a <= 20 , c'est à dire si a]10,20]
(a > 10) **||** (b <= 20) sera vrai si a > 10 ou si b <= 20
!(a > 10) sera vrai si a <= 10

V. Les instructions élémentaires

1. L'affectation

- Cette instruction permet de donner la valeur de l'expression de droite à la variable de gauche.
- Le langage C utilise le symbole **=** pour l'affectation.
- Pour affecter un **caractère**, il faut l'entourer d'**apostrophes** (sinon le compilateur pourrait le confondre avec un nom de variable). Pour une chaîne, il faut utiliser des **guillemets**.

Exemples :

x = 5;
La variable x prend pour valeur 5

i = i + 1;
La variable i prend pour valeur son ancienne valeur augmentée de 1

C = 'a';
La variable C prend pour valeur le caractère 'a'.

V. Les instructions élémentaires

2. L'affichage

- La fonction **printf** permet d'afficher des informations à l'écran

  Syntaxe: **printf (" chaîne de caractères " , variable1, variable2, ...);**

- Cette fonction, contenue dans la bibliothèque standard **stdio.h**, attend comme premier paramètre la chaîne de caractère à afficher avec éventuellement la description des formats d'affichage des variables à afficher dans cette chaîne : Les paramètres suivants correspondent, dans l'ordre, à chaque variable dont on souhaite afficher la valeur.

- Les formats d'affichage sont:
  - **%d** ou **%ld** pour les entiers (int, short, long)
  - **%f** ou **%lf** pour les réels (float, double)
  - **%s** pour les chaînes de caractères
  - **%c** pour les caractères

  Exemples:
  - **printf("La valeur des variables X et Y sont : %d et %d",X,Y) ;**

    Les deux %d seront respectivement remplacés à l'écran par la valeur de X et de Y.
  - **printf ("%d", y - x);**

    Cette instruction affiche 5 si y=10 et x=5
  - **printf ("la valeur de x est %d et celle de y est %d", x, y);**

    En supposant que x ait pour valeur 5 et y pour valeur 10, on aura à l'écran La valeur de x est 5 et celle de y est 10

V. Les instructions élémentaires

2. L'affichage (sauts de ligne)

- L'instruction **printf ("Bonjour\nCa va?") ;** va donner à l'écran :

```
Bonjour
Ca va?
```

- Les instructions:

```
int i =1;
float x = 2.0 ;
printf (" Bonjour \ n ");
printf (" i = %d \ n " , i);
printf (" i = %d , x = % f \ n " , i , x);
```

Donne à l'écran

```
Bonjour
i = 1
i = 1, x=2.0
```

V. Les instructions élémentaires

3. La lecture

- L'instruction **scanf** permet au programme de lire des informations saisies au clavier par l'utilisateur.

  Syntaxe : **scanf(" chaîne de formatage" , &variable1 , &variable2, . . .)**

- Cette fonction, également contenue dans la bibliothèque standard stdio.h, attend comme premier paramètre la chaîne décrivant les formats de lecture des variables à lire au clavier. Les paramètres suivants sont, dans l'ordre, l'adresse des variables dont on souhaite lire la valeur.

- **Formats de lecture :**
  - %d ou %ld pour les entiers (int, short, long)
  - %f ou %lf pour les réels (float, double)
  - %s pour les chaînes de caractères
  - %c pour les caractères

  Exemple: **scanf(" %d %d",&X,&Y) ;**
  Le **&** permet d'accéder à l'adresse des variables. (A omettre dans le cas de la lecture d'une chaîne de caractères).

- **scanf** lit au clavier les valeurs introduites successivement.
  Exemple: **scanf ("%d %d",&a ,&b);**
  Si l'on écrit au clavier 2 puis 3, la variable a va prendre pour valeur **a** 2 et **b** va la valeur 3.

V. Les instructions élémentaires

3. Lecture et affichage de caractères

- Les fonctions **getchar** et **putchar** de la bibliothèque **stdio.h** permettent respectivement au programme de lire au clavier et d'afficher à l'écran des caractères. Il s'agit de fonctions d'entrées-sorties non formatées.

- La fonction getchar permet au programme de lire un caractère saisi au clavier par l'utilisateur. Cette fonction retourne un entier (code ASCII) correspondant au caractère lu. Pour mettre le caractère lu dans une variable c de type caractère, on écrit

  c = getchar() ;

- La fonction putchar permet d'afficher un caractère à l'écran. La syntaxe pour afficher un caractère c à l'écran est la suivante :

  putchar(c) ;

Remarque: La lecture avec la fonction getchar se fait par l'intermédiaire d'une zone mémoire appelée "buffer d'entrée" qui sert à contenir les caractères tapés au clavier par l'utilisateur et qui fonctionne selon le principe d'une "file" (premier arrivé = premier sorti).

- Si le buffer d'entrée n'est pas vide, l'instruction getchar(); lit le caractère le plus ancien se trouvant dans le buffer d'entrée sans attendre l'utilisateur, et enlève ce caractère du buffer d'entrée. Si le buffer d'entrée est vide, l'instruction getchar(); stoppe le déroulement du programme jusqu'à ce que le buffer d'entrée reçoive un ou plusieurs caractères. L'ajout de caractères dans le buffer d'entrée ne se fait qu'au moment où l'utilisateur tape un retour-chariot (touche Entrée). Pour vider le buffer d'entrée, il faut utiliser l'instruction fflush(stdin);

- Certains compilateurs proposent une bibliothèque appelée **conio.h** qui contient deux fonctions de lecture et d'écriture de caractère. Ce sont respectivement les fonctions **getch** et **putch**.

  c = getch( ) ;
  putch(c) ;
  La fonction getch lit directement le caractère tapé par l'utilisateur sans utiliser de buffer d'entrée. Ces deux fonctions ne font pas partie du langage C standard.