Le langage C - Les bases de la syntaxe en C

Voici les choses essentielles à connaître sur la syntaxe du C.

5TTR 6TTR Découverte

Objectifs

À la fin de ce chapitre, tu sauras :

- Comprendre la structure minimale d'un programme en C.
- Exécuter ton premier programme avec `printf`.
- Expliquer le rôle des accolades `{}` et du point-virgule `;`.
- Utiliser des commentaires pour documenter ton code.
- Compiler et exécuter un programme en C.

Structure minimale d'un programme

En C, tout programme contient **au minimum** :

```c
#include <stdio.h>   // bibliothèque pour afficher du texte

int main() {
    return 0; // indique que le programme s'est bien terminé
}
```

- `#include <stdio.h>` : permet d'utiliser la fonction `printf`.
- `int main() { ... }` : le point d'entrée du programme (là où tout commence).
- `return 0;` : dit au système que le programme s'est terminé correctement.

2 Les instructions et les blocs

- Une instruction se termine toujours par un **point-virgule** `;`.

- Les **accolades** `{ }` regroupent plusieurs instructions en bloc (ex. dans `main`).

Sans le `;` ou sans les `{}`, ton programme ne compilera pas.

Les commentaires

Un commentaire sert à expliquer ton code. Le compilateur les ignore.

- Une ligne :

```
// Ceci est un commentaire
```

- Plusieurs lignes :

```
/* Ceci est
   un commentaire
   sur plusieurs lignes */
```

Compilation et exécution

Contrairement à Python, le C **doit être compilé** avant de s'exécuter.

- Étape 1 : Écrire ton fichier `hello.c`.
- Étape 2 : Compiler avec `gcc hello.c -o hello`
- Étape 3 : Lancer le programme avec `./hello`

Exemple pratique

```c
#include <stdio.h>

int main() {
    printf("Bonjour, je découvre le C !\n");
    printf("Je peux afficher plusieurs lignes.\n");
    return 0;
}
```

Exercices

1. Crée un programme qui affiche ton prénom.
2. Ajoute un second `printf` qui dit ton âge.
3. Mets un commentaire expliquant chaque ligne.
4. Supprime un `;` volontairement : que se passe-t-il ?
5. Supprime les `{}` et compile : que se passe-t-il ?

À retenir

- Un programme C commence toujours par `int main()`.
- Les instructions se terminent par `;`.
- Les `{}` regroupent un bloc d'instructions.
- Les commentaires servent à documenter ton code.
- Le C doit être compilé avant de s'exécuter.

Parfait 👍 Tu as raison : c'est un point essentiel à souligner pour bien marquer la différence. Voici la version mise à jour de l'introduction (modifiée dans la partie **compilé vs interprété** et **à retenir**).

Introduction au langage C

Objectifs

À la fin de ce chapitre, tu sauras :

- Situer le langage C dans l'histoire de l'informatique.
- Expliquer la différence entre un langage **compilé** et un langage **interprété**.
- Comprendre pourquoi on apprend encore le C aujourd'hui.
- Connaître les outils nécessaires pour programmer en C.

Le C en quelques mots

Le langage C a été créé dans les années 1970.

- Il a servi à écrire **le système Unix** et plus tard **Linux**.
- Beaucoup de logiciels systèmes (compilateurs, bases de données, microcontrôleurs) utilisent encore le C.
- Le C est un langage **rapide** et **proche de la machine** : il permet de comprendre comment l'ordinateur gère la mémoire et exécute les instructions.

Langages compilés et interprétés

Il existe deux grandes familles de langages :

- **Langages interprétés** (ex. Python, JavaScript)
  - Ton code est **lu ligne par ligne** par un interpréteur.
  - Tu dois avoir cet interpréteur installé pour lancer ton programme.
    - Exemple : `python moncode.py` → il faut Python installé.
  - Avantage : facile à tester rapidement.
  - Inconvénient : plus lent, dépend toujours de l'interpréteur.
- **Langages compilés** (ex. C, C++)
  - Ton code est **traduit en langage machine** avant d'être exécuté → création d'un fichier exécutable.
  - Une fois compilé, ton programme peut tourner **sans avoir besoin du compilateur**.
    - Exemple : `moncode.c` → compilation → `moncode.exe`, exécutable directement.
  - Avantage : programme **rapide et autonome**.
  - Inconvénient : il faut passer par l'étape de **compilation**.

💡 **Comparaison rapide :**

- Python → `python moncode.py` (l'interpréteur est indispensable).
- C → `gcc moncode.c -o moncode` puis `./moncode` (le programme compilé peut être exécuté seul).

Pourquoi apprendre le C

- Comprendre **comment fonctionne un ordinateur** en profondeur.

- Apprendre la rigueur d'un langage plus « strict » que Python ou JavaScript.
- Utile pour la **programmation système**, l'**embarqué** (microcontrôleurs, robots), et même pour certains moteurs de jeux vidéo.

Les outils à utiliser

Pour écrire et lancer un programme en C, tu as besoin de :

1. **Un éditeur de code / IDE** : Code::Blocks, Visual Studio Code, CLion…
2. **Un compilateur** : le plus courant est `gcc` (GNU Compiler Collection).
3. **Un terminal / console** pour compiler et exécuter ton programme.

👉 On fera un article spécifique pour t'expliquer **comment installer et utiliser ces outils**.

Exercices de réflexion

1. Classe ces langages dans la bonne catégorie : C, JavaScript, Python, C++.
2. Pourquoi dit-on qu'un programme compilé est plus rapide qu'un programme interprété ?
3. Imagine : tu veux programmer un petit jeu sur ton navigateur. Choisirais-tu plutôt C ou JavaScript ? Pourquoi ?
4. Explique pourquoi tu ne peux pas lancer directement `moncode.c` comme un fichier exécutable.

À retenir

- Le C est un langage **compilé**, contrairement à Python ou JavaScript qui sont **interprétés**.
- Un langage interprété → nécessite **toujours l'interpréteur installé** pour exécuter ton programme.
- Un langage compilé → produit un **exécutable autonome**, que tu peux lancer directement.
- Le C reste aujourd'hui incontournable pour comprendre **l'informatique au plus près de la machine**.

Veux-tu que je mette aussi un **schéma visuel simple** (pipeline "Code source → Interpréteur → Résultat" vs "Code source → Compilation → Exécutable → Résultat") pour renforcer la compréhension des élèves ?