

Introduction

Programmation impérative

Emna Chebbi, Revekka Kyriakoglou

Plan du cours

- 1 Objectifs
 - Ce cours
 - Evaluation
- 2 Pourquoi le C ?
- 3 Mise en place
- 4 Les fichiers sources
- 5 Compilation
- 6 Hello word
- 7 Bonnes pratiques
- 8 printf() et scanf()
- 9 Erreurs et avertissements

Objectifs

L'objectif de ce cours est d'apprendre le langage C :

- produire du code propre,
- gestion de la mémoire,
- pointers,
- déboguer un programme C.

Éléments importants :

- Vous avez accès à un ordinateur.
- Vous savez comment utiliser le web pour trouver des informations sur les sujet dont vous avez besoin.
- Vous êtes familiarisé avec l'utilisation du terminal.
- **Vous voulez apprendre à programmer en C!!!**

Evaluation

Ce cours sera noté comme suit :

- 1 contrôle soutenu
- 2 petites interrogations
- 3 projet final

Pourquoi le C ?

Programmation impérative : Le développeur écrit un code qui spécifie les étapes que l'ordinateur doit suivre pour atteindre l'objectif.

- Syntaxe relativement simple, reprise par de nombreux autres langages.
- Il vous aidera à comprendre comment fonctionne un ordinateur (comme l'allocation et la gestion de la mémoire).
- Il est un langage structuré, qui permet de décomposer un programme complexe en programmes plus simples appelés fonctions.
- La plupart des langages de programmation peuvent s'interfacer avec C (ex. Cython).
- Executables rapides.

Mise en place

Vous avez besoin de :

- compilateur,
- un endroit pour mettre vos programmes.



Pour les systèmes d'exploitation Linux, Unix et Mac OS, le compilateur C **est déjà inclus**.



Si vous utilisez une machine Windows, vous devez télécharger et installer un compilateur GCC pour Windows.

Créer un répertoire **prog/c/cours**

- 1 Ouvrir le terminal dans le répertoire HOME.



Si vous n'y êtes pas, tapez **cd** pour y retourner.

- 2 Créer le répertoire **prog/c/cours**



mkdir -p prog/c/cours

Editeur

Vous pouvez utiliser l'éditeur de votre choix.

Voici quelques exemples :

- Gedit
- Visual studio code
- Sublime Text
- Eclipse
- Atom

Les fichiers sources

- Le code en C doit être contenu dans un fichier dont le nom se termine par **.c**.
- Les fichiers header terminent par **.h** et ils contiennent les déclarations de fonction C et les macro définitions à partager entre plusieurs sources. (nous les verrons plus tard)

Un fichier source écrit en langage C est composé de :

- Définition de types.
- Déclaration de variables.
- Déclaration de fonctions.



Syntaxe très précise à respecter !

Compilation

Un fichier source C est du texte, ce ne sont pas des instructions que le processeur peut exécuter. Plutôt que de traduire à la main le source C en instructions pour le processeur, il est plus pratique d'utiliser un outil : **le compilateur C**.



Le compilateur C est accessible à travers la commande **gcc**.



Compilation

```
! gcc -Wall hello.c -o Helloworld
```



-Wall demande au compilateur d'afficher tous les messages de prévention.



L'option **-o** de gcc demande la compilation en précisant le nom à donner à l'exécutable.

Hello word !

```
1  /*Affichage du message Hello world ! */
2
3  #include<stdio.h>
4
5  int main(void){
6  printf("Hello_world_!\n");
7  return 0;
8  }
```

Hello word !

```

1  /*Affichage du message Hello world ! */
2
3  #include<stdio.h>
4
5  int main(void){
6  printf("Hello_world!\n");
7  return 0;
8  }

```

- On met les commentaires entre `/* ... */`.
- Les lignes qui commencent par un dièse sont des **directives pour le préprocesseur**. Elle sont exécutées lors d'une première phase de traduction. Ici on demande au compilateur l'inclusion de la bibliothèque standard du C **stdio.h** permettant l'utilisation de `printf`.
- Les lignes 5-8 sont ce qu'on appelle une **fonction**. C'est la fonction qui sera exécutée au lancement du programme.

Hello word !

```

1  /*Affichage du message Hello world ! */
2
3  #include<stdio.h>
4
5  int main(void){
6  printf("Hello_world_!\n");
7  return 0;
8  }
```

- Le int en ligne 5 est le type de retour de la fonction.
- Les **parametres** de la fonction main sont précisés dans la parenthèse (ligne 5). Ici la fonction ne reçoit rien (void).
- Les lignes à l'intérieur d'une fonction sont appelées **instructions**.
- Chaque programme une fois terminé renvoie une valeur, par exemple pour dire que tout s'est bien passé.



Avec un type de retour void on utilise **return ;** ("retourner rien").

Hello word !

```
1  /*Affichage du message Hello world ! */
2
3  #include<stdio.h>
4
5  int main(void){
6  printf("Hello_world_!\n");
7  return 0;
8  }
```

- L'accolade ouvrante { debute un bloc d'instructions.
- L'accolade fermée } termine un bloc d'instructions.
- La fonction **printf** affiche a l'écran la chaîne de caractère souhaitée.
Voici son prototype :
int printf(const char format, ...);*



Il est nécessaire d'inclure l'en-tête standard **<stdio.h>** au début du code source du programme, car c'est lui qui permet de déclarer la fonction printf.

int main() VS int main(void)

```
//Programme 1
int main(){
    //qqch
    return 0;
}
```

```
//Programme 2
int main(void){
    //qqch
    return 0;
}
```



Les deux définitions fonctionnent également en C, mais **la deuxième définition avec void est considérée comme meilleure** car elle spécifie clairement que main ne peut être appelé que sans aucun paramètre.

function(void) VS fonction()

//Programme 1

```
void f(){
    return;
}
int main(void){
    f(100);
    return 0;
}
```

//Programme 2

```
void f(void){
    return;
}
int main(void){
    f(100);
    return 0;
}
```

Le premier programme se compile mais pas le deuxième.

error : too many arguments to function 'f'

Return



Une bonne pratique consiste à toujours spécifier un type de retour pour vos fonctions.



Si une valeur de retour n'est pas nécessaire, déclarez que la fonction a un type de retour **void**.



Si un type de retour n'est pas spécifié, le compilateur C suppose que le type de retour par défaut est **int**.

printf() et scanf()



printf()

La fonction printf() est utilisée pour transférer du texte, des valeurs de variables ou des résultats d'expressions vers le fichier de sortie standard stdout (par défaut l'écran).

```
printf("valeur de x est : %d",x);
```



scanf()

La fonction scanf() est utilisée pour lire un entier au clavier et placer la valeur à l'adresse d'une variable.

Il faut fournir à scanf l'adresse de la variable qui reçoit la valeur (on utilise "&")

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

Erreur et avertissement



Error

Error est suivi d'un message concernant la nature de l'erreur détectée. Ce qui signifie que la compilation a échoué en un point du programme source. L'erreur se situe donc avant ce point.



Warning

Warning est suivi d'un message qui est un avertissement. Le compilateur a réalisé le travail mais vous signale qu'il a détecté un problème potentiel quand vous exécuterez votre programme.



Votre code ne doit pas générer de message d'erreur !



Un code bien écrit n'a pas non plus de message « warning » !

Types d'erreurs

Il existe principalement cinq types d'erreurs dans la programmation C :

- Syntax error
- Run-time error
- Linker error
- Logical error
- Semantic error

- **Syntax error** : Ces erreurs sont principalement dues à des fautes de frappe ou au non-respect de la syntaxe du langage de programmation spécifié. Une telle erreur se produit si le point-virgule (;) est absent à la fin de la déclaration.
- **Run-time error** : Lorsque le programme est en cours d'exécution, et qu'il n'est pas en mesure d'effectuer l'opération. La division par zéro est un exemple courant de cette erreur.
- **Linker error** : Ces erreurs sont principalement générées lorsque le fichier exécutable du programme n'est pas créé. Cela peut être dû à un mauvais prototypage de la fonction ou à l'utilisation d'un mauvais fichier .h. Une telle erreur consiste à utiliser Main() au lieu de main().
- **Logical error** : L'erreur logique est une erreur qui conduit à une sortie non souhaitée.
- **Semantic error** : Les erreurs sémantiques sont les erreurs qui se produisent lorsque les déclarations ne sont pas compréhensibles par le compilateur. L'utilisation d'une variable non initialisée produit une telle erreur.



Exercice 1

Essayez de compiler le code suivant. Quelle erreur voyez-vous ?
Comment la corriger ?

```
1  /*Affichage la valeur de a ! */
2
3  #include <stdio.h>
4
5  int main(void){
6  int a = 10;
7  printf("La_valeur_de_a_est_:_%d\n",a);
8  return;
9  }
```



Exercice 2



Créer votre premier programme qui affichera la date à l'écran.

Quiz

Question 1 : Quel est le nom de la fonction principale d'un programme ?

- 1 printf
- 2 main
- 3 princ

Question 2 : Quel est le nom de la fonction permettant d'afficher du texte à l'écran ?

- 1 print
- 2 imprimf
- 3 printf

Question 3 : Qu'est-ce qu'une bibliothèque ?

- 1 Un fichier source déjà écrit contenant des fonctions toutes prêtes
- 2 Un fichier permettant d'exécuter
- 3 Un lieu où est conservée et lue une collection organisée de livres.