Année académique : 2024/2025
2IDL - FS Tétouan

Généralités sur le langage C

Introduction et instructions de base

Pr. Zaynab El Khattabi

Premiers pas…

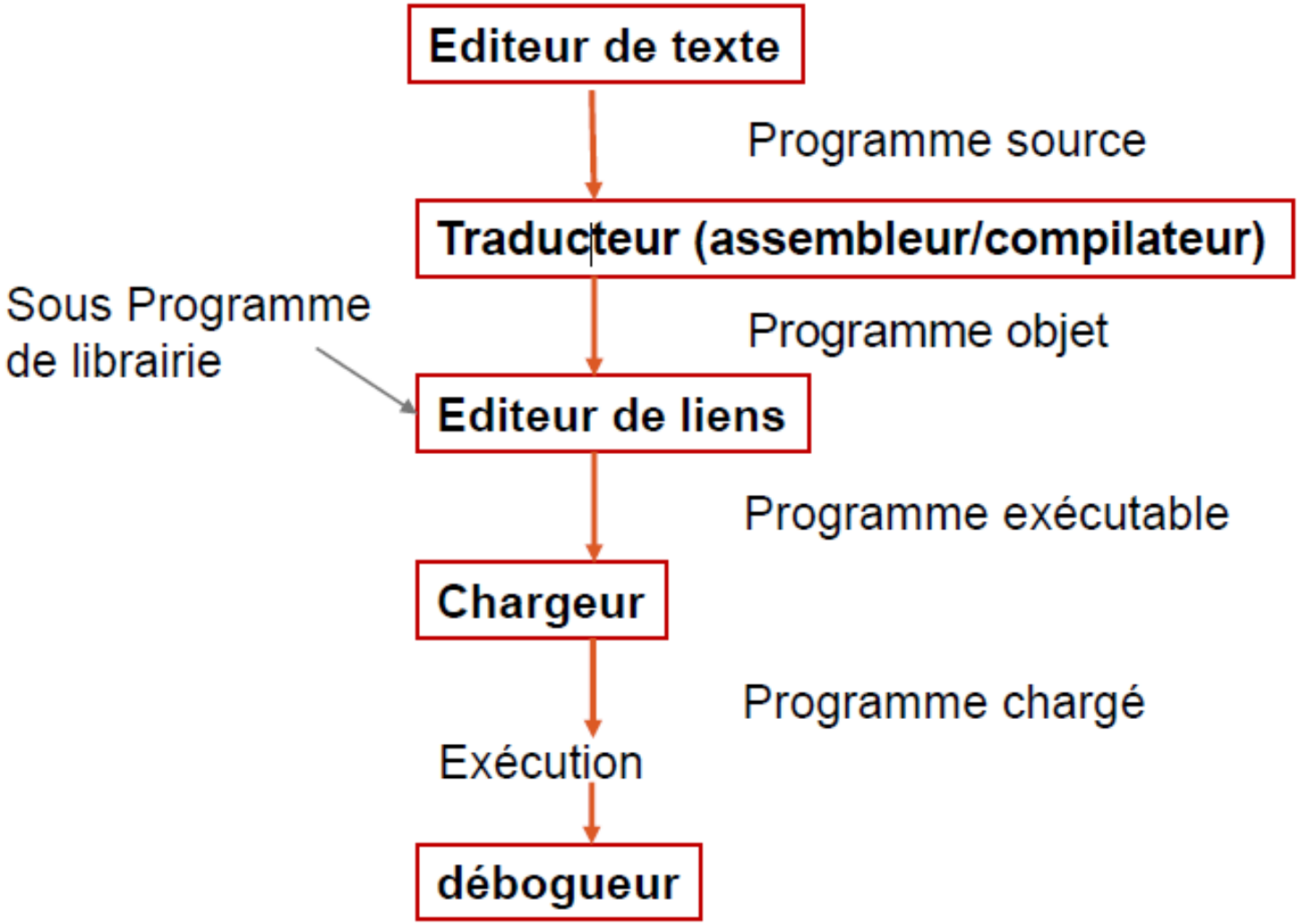
Qu’est-ce qu’un programme ?

- Un programme est une séquence d’**instructions**, d’ordres, donnés à notre ordinateur afin qu’il exécute des actions.

- Ces instructions sont exécutées par un composant précis de l’ordinateur : **le processeur.**

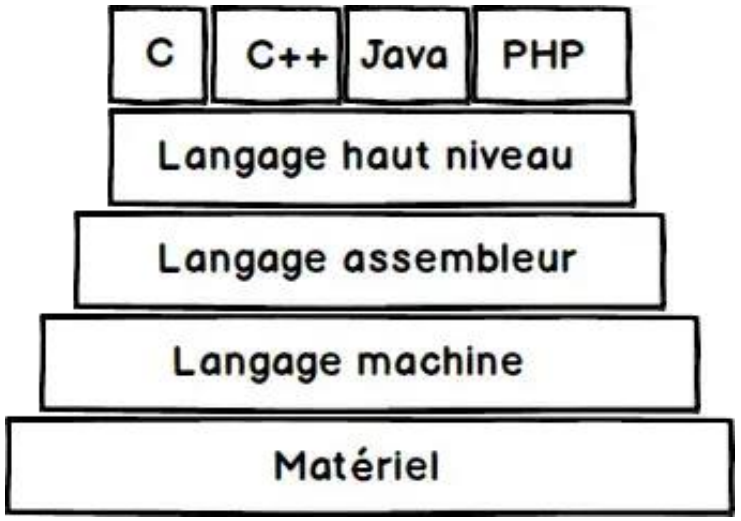
Premiers pas…

Étapes de préparation d'un programme



Langages de programmation

Niveaux de langage

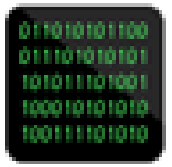


Langage de haut niveau
Plus simple et plus éloigné du fonctionnement de la machine

- Java
- C# .NET
- Python
- Ruby...

Langage de bas niveau
Plus complexe et plus proche du fonctionnement de la machine

- C
- C++
- Objective-C...



- Assembleur

Binaire

Langages de programmation

Langage haut niveau

- Les langages de programmation indépendants du matériel sont appelés des langages haut niveau.

- Aucune connaissance particulière du matériel n’est nécessaire, car les langages haut niveau créent des programmes portables et non liés à un ordinateur ni à une puce.

- Quelques exemples du langage haut niveau sont C, C++, Java, PHP, etc.

Langages de programmation

Langage bas niveau

- Les langages bas niveau sont utilisés pour écrire des programmes relatifs à l'architecture et au matériel spécifiques d'un type d'ordinateur particulier.

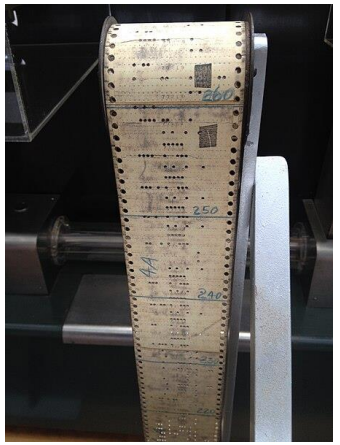
- Ils sont plus proches du langue maternelle d'un ordinateur (le binaire), ce qui les rend plus difficiles à comprendre pour les programmeurs.

- Un exemple du langage bas niveau est assembleur, …

Un peu d'histoire

- La programmation se faisait directement en langage machine, au début à l'aide de cartes ou bandes perforées, puis à l'aide de consoles.

- Avec l'amélioration progressive et rapide des ordinateurs, vers la fin des années 1950, il devient possible d'envisager des solutions plus efficaces.
  - Un des problèmes était celui du stockage.



- Une fois cette contrainte disparue, d'autres approches deviennent envisageables:
  - Programmer dans **un autre langage** qui **sera traduit en langage machine**.

- Un premier programme de « traduction » est écrit en langage machine et est ensuite utilisé pour traduire un autre langage vers le langage machine. Un de ces premiers langages est **l'Assembleur.**

Un peu d’histoire
Le Langage C

- Le langage C est né **au début des années 1970** dans les laboratoires de la société **AT&T** aux **États-Unis**.

- Son concepteur, **Dennis MacAlistair Ritchie**, souhaitait améliorer un langage existant, le B, afin de lui adjoindre des nouveautés.

- Son succès fut tel auprès des informaticiens qu’en 1989**, l’ANSI**, puis **en 1990, l’ISO**, décidèrent de le normaliser, c’est-à-dire d’établir des règles internationales et officielles pour ce langage.

Python vs C

	C	Python
Syntaxe	Plus complexe Moins d'erreurs de typographie Utilise accolades, points virgules	Plus lisible Utilise l'indentation, deux points
Portabilité	Bonne portabilité	Meilleure portabilité Multi-platforme Moins de modifications nécessaires
Performance	Compilé Détecte erreurs dès la compilation Plus efficaces pour calculs intensifs et gestion de données volumineuses	Interpreté Plus facile pour les tâches de script
Bibliothèques	Plus grande flexibilité pour les taches de bas niveau	Plus vastes collections de bibliothèques nécessaires pour les projets

Programmer en C

Pour programmer en C:

1. **Un éditeur de texte :** l'écriture du code source.
   - Techniquement, n'importe quel éditeur de texte suffit, mais il est souvent plus agréable d'en choisir un qui n'est pas trop minimaliste ;
2. **Un compilateur :** GCC **,**permettre de transformer le code écrit en langage C en un fichier exécutable.

- Il existe deux solutions : utiliser ces deux logiciels séparément ou bien les utiliser au sein d'un **environnement intégré de développement** (abrégé EDI).

Premier programme C

1. Editer le script
2. L'enregistrer avec l'extension **.c** (par exp **main.c**)
3. Compiler avec le compilateur **GCC**

   `gcc -Wall -std=c11 main.c`
4. Exécuter en tapant:

   ./main.exe ou **./main.out**.

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

- La fonction **main()** ne comporte qu'une seule instruction : **return 0;** , qui met fin à son exécution (et, par conséquent, à celle du programme) et permet d'indiquer au système d'exploitation que l'exécution s'est correctement déroulée (**une valeur différente de zéro indiquerait une erreur)**.

Premier programme C

Les commentaires

- Il est possible d'ajouter des commentaires dans un code source, par exemple pour décrire des passages un peu moins lisibles ou tout simplement pour offrir quelques compléments d'information au lecteur du code.

```
int main(void)
{
// Ceci est un commentaire.

/* Ceci est un autre commentaire. */

/* Ceci est un commentaire qui
   prends plusieurs lignes. */

    return 0;
}
```

Notions de base

- Les variables
- Manipulations basiques des entrées/sorties
- Les opérations mathématiques
- Tests et conditions
- Les sélections
- Les boucles
- Les sauts
- Les fonctions

Les variables
Déclaration

- Une variable correspondra à une portion de mémoire à laquelle nous donnerons un nom. Ce nom permettra d'identifier notre variable, tout comme une référence permet d'identifier une portion de mémoire parmi toutes les autres.
- Une variable est constituée de deux éléments obligatoires :
  - un **type** ;
  - un **identificateur** qui est en gros le « nom » de la variable.

- Pour déclarer une variable:

```
type identificateur;
```

Les variables types

- Un type permet d'indiquer au compilateur quel genre de données nous souhaitons stocker. Ce type va permettre de préciser :
  - toutes les valeurs que peut prendre la variable ;
  - les opérations qu'il est possible d'effectuer avec.

- Le langage C fournit dix types de base.

Type	Pour stocker
_Bool	un entier
char	un caractère
signed char	un entier
short int	un entier
int	un entier
long int	un entier
long long int	un entier
float	un réel
double	un réel
long double	un réel

Les variables
Types

Tailles des types

- **vide** : void . Aucune variable ne peut être de ce type.

- **entiers**, par taille-mémoire croissante :
  - *char*, stocké sur un octet ; valeurs : de −27 à 27 − 1 (−128 à 127),
  - *short*, stocké sur 2 octets ; valeurs : de −215 à 215 − 1 (−32768 à 32767),
  - *long*, stocké sur 4 octets ; valeurs : de −231 à 231 − 1,
  - *int*, coïncide avec short ou long, selon l'installation.

- **réels**, par taille-mémoire croissante :
  - *float*, stocké sur 4 octets ; précision : environ 7 chiffres,
  - *double*, stocké sur 8 octets ; précision : environ 15 chiffres,
  - *long double*, stocké sur 10 octets ; précision : environ 18 chiffres.

Les variables

- Toute variable doit être **définie** avant d'être **utilisée** !
- Une définition peut apparaître n'importe ou dans un programme.
- Une variable est définie jusqu'à la fin de la première instruction composée (marquée par **}**) qui contient sa définition.
- (Une variable définie en dehors de toute fonction – et de tout espace de nom – est une **variable globale**).

Les variables
Identificateurs

- Un identificateur est un nom donné à une variable pour la différencier de toutes les autres. Et ce nom, c'est au programmeur de le choisir. Cependant, il y a quelques limitations à ce choix.

  - Seuls les 26 lettres de l'alphabet latin (majuscules ou minuscules), le trait de soulignement « _ » et les chiffres sont acceptés. Pas d'accents, pas de ponctuation ni d'espaces ;
  - Un identificateur ne peut pas commencer par un chiffre ;
  - Les mots-clés ne peuvent pas servir à identifier une variable ;

- À noter que le C fait la différence entre les majuscules et les minuscules (**on dit qu'il respecte la casse**). Ainsi les trois identificateurs suivants sont différents. (variable, Variable, VaRiAbLe)

Les variables
Identificateurs

Exemples: Déterminer les identificateurs correctes et incorrectes.

- variable
- Nom de variable
- 1nombre_de_vie
- test!
- nombre_de_vie
- continue
- un_dernier_pour_la_route1

Les variables

Exemples de déclaration de variables

```
int main(void)
{
    _Bool var_boolean;
    double taille;
    unsigned int age;
    char caractere;
    short petite_valeur;

}
```

Les variables

- Il est possible de déclarer plusieurs variables de même type sur une même ligne, en séparant leur noms par une virgule.

```
int age, annee, mois;
```

- Il est possible d'initialiser une variable, c'est-à-dire de lui attribuer une valeur. La syntaxe est la suivante:

```
type identificateur = valeur;
```

- Exemples:

```
_Bool booleen = 0;
unsigned char age = 42;
long score = -1;
```

Les variables

- **Initialisation des types réels**

```
double pi = 3.14;
```

  - **En notation scientifique**

```
double x = 1E-1;
```

- **Initialisation du type char**

```
char a = 'A';
```

- **Déclarer une constante**

```
const double pi = 3.14159265;
```

Les variables
L'affectation

- L'affectation permet de modifier la valeur contenue dans une variable pour la remplacer par une autre valeur.
- Il n'y a aucune limite au nombre d'affectations:

```
a=3;
a=0;
a=4;
```

- **Le code suivant est correcte?? Pourquoi?**

```
int age = 15;
int age = 20;
```

- Sans initialisation ou affectation, la valeur d'une variable est **indéterminée.**

Manipulations basiques des entrées/sorties

- Durant l'exécution d'un programme, le processeur a besoin de communiquer avec le reste du matériel.

- Ces échanges d'informations sont les **entrées** et les **sorties** (ou **input** et **output**), souvent abrégées **E/S** (ou **I/O**).

- Les entrées permettent de recevoir une donnée en provenance de certains périphériques.

- À l'inverse, les sorties vont transmettre des données vers ces périphériques.

Manipulations basiques des entrées/sorties

Les sorties:

Exemple d'affichage d'un texte sur la console

```c
#include <stdio.h>


int main(void)
{
    printf("Bonjour tout le monde !\n");
    return 0;
}
```

Manipulations basiques des entrées/sorties

Les sorties:

Exemple d'affichage d'un texte sur la console

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

- Il s'agit d'une **directive** du préprocesseur.
- Celle-ci sert à inclure un fichier (« **stdio.h** ») qui contient les références de différentes fonctions d'entrée et sortie.
- Un fichier se terminant par l'extension « **.h** » est appelé un fichier d'en-tête (**header**) et fait partie avec d'autres d'un ensemble plus large appelée **la bibliothèque standard**.
- Le **\n** est un caractère spécial qui représente un retour à la ligne.

Manipulations basiques des entrées/sorties

Les sorties – les formats:

- La fonction printf() met à disposition des formats.
- Ceux-ci sont des sortes de repères au sein d'un texte qui indiquent à **printf()** que la valeur d'une variable est attendue à cet endroit.
- **Exemple:**

```c
#include <stdio.h>


int main(void)
{
    int var = 30;

    printf("%d\n", var);
    return 0;
}
```

Manipulations basiques des entrées/sorties

Liste des formats:

- Un format commence toujours par le symbole **%**

Type	indicateur
_Bool	d (ou i)
char	c
signed char	d (ou i)
short	d (ou i)
int	d (ou i)
long	ld (ou li)
long long	lld (ou lli)
unsigned char	u, x (ou X) ou o
unsigned short	u, x (ou X) ou o
unsigned int	u, x (ou X) ou o
unsigned long	lu, lx (ou lX) ou lo
unsigned long long	llu, llx (ou llX) ou llo
float	f, e (ou E) ou g (ou G)
double	f, e (ou E) ou g (ou G)
long double	Lf, Le (ou LE) ou Lg (ou LG)

Manipulations basiques des entrées/sorties

```c
#include <stdio.h>
int main(void)
{  char z = 'b';
    char a = 1;
    unsigned short b = 20;
    int c = 30;
    long d = 40;
    float e = 50.;
    double f = 60.0;

    printf("%c\n", z);
    printf("%d\n", b);
    printf("%u\n", b);
    printf("%d\n", c);
    printf("%li\n", d);
    printf("%f\n", e);
    printf("%e\n", f);

    return 0;
}
```

Manipulations basiques des entrées/sorties

Les sorties – les caractères spéciaux:

Séquence d'échappement	signification
\a	Caractère d'appel
\b	Espacement arrière
\f	Saut de page
\n	Saut de ligne
\r	Retour chariot
\t	Tabulation horizontale
\v	Tabulation verticale
\"	Le symbole « " »
\\	Le symbole « \ » lui-même

Manipulations basiques des entrées/sorties

Les sorties – les caractères spéciaux - Exemples:

```c
#include <stdio.h>


int main(void)
{
    printf("Quelques sauts de ligne\n\n\n");
    printf("\tIl y a une tabulation avant moi !\n");
    printf("Un message à écraser...\r");
    printf("Nouveau message    !\n");
    return 0;
}
```

Manipulations basiques des entrées/sorties

L’entrée:

- Récupérer l’information d’entrée grâce à la fonction **scanf(),** dont l’utilisation est assez semblable à printf().

- La fonction scanf() a besoin de connaître l’emplacement en mémoire de nos variables afin de les modifier. Afin d’effectuer cette opération, nous utilisons le symbole **&.**
  - concept de transmission d’adresses mémoires.

- Il est possible de lire plusieurs entrées en même temps

Manipulations basiques des entrées/sorties

L'entrée – Exemples:

```
#include <stdio.h>
int main(void)
{
    int num;
    printf("Saisir un numero ? ");
    scanf("%d", &num);
    printf("Vous avez saisi: %d \n", num);
    return 0;
}
```

Manipulations basiques des entrées/sorties

L'entrée – Exemples:

```
#include <stdio.h>
int main(void)
{
int a, b;
scanf("%d %d", &a, &b);
printf("a = %d , b = %d\n", a, b);
return 0;
}
```

Opérations mathématiques

- Le C fournit des opérateurs pour réaliser les opérations mathématiques de base
- La priorité des opérateurs est identique à celle décrite en mathématiques
- En cas de mélange des types lors d'une opération, le C prévoit des règles de conversions
- Il est possible de convertir un opérande d'un type vers un autre, certaines conversions sont implicites.

Opérations mathématiques

Opérateurs arithmétiques

- **+** : addition,
- **-** : soustraction,
- ***** : multiplication,
- **/** : division. entre deux entiers, donne le quotient entier, (19 / 5 vaut 3)
- **%** : entre deux entiers, donne le reste modulo. (19 % 5 vaut 4)

Opérateurs de comparaison

< (inférieur), **<=** (inférieur ou égal), **==** (égal), **>** (supérieur), **>=** (supérieur ou égal) et **!=** (différent)

Opérateurs booléens

&& représente l'opérateur "**ET**", **||** représente le "**OU**", et **!** représente le "**NON**".

Opérations mathématiques

Exemples: Opérations sur les variables et les constantes

```c
int main(void)
{
int a = 1;
int b = 9;
int somme = a + b;

printf("%d + %d = %d\n", a, b, somme);

    return 0;
}
```

```c
int main()
{

int a = 5;
int b = 65;

printf("%d\n", b / a * 2 + 7 % 2);

}
```

Opérations mathématiques

++ : incrémentation. Si i est de type entier, les expressions i++ et ++i ont toutes deux pour valeur la valeur de i. Mais elles ont également un *effet de bord* qui est :

- pour la première, d'ajouter **ensuite 1** à la valeur de i (***post-incrémentation***),
- pour la seconde, d'ajouter **d'abord 1** à la valeur de i (***pré-incrémentation***).

-- : décrémentation. i-- et --i fonctionnent comme i++ et ++i, mais retranchent 1 à la valeur de i au lieu d'ajouter 1.

Opérations mathématiques

Exemple:

```c
int main()
{
    int x = 1;
    int y = 1;
    int a = x++;
    int b = ++y;

    printf("a = %d\n", a);
    printf("b = %d\n", b);
    printf("x = %d\n", x);
    printf("y = %d\n", y);
}
```

Résultat d'exécution:

a=1
b=2
x=2
y=2

Opérations mathématiques

Les opérateurs combinés

- Il existe des opérateurs combinés qui réalisent une affectation et une opération en même temps.

```c
int main()
{

int ma_var = 4;
ma_var = ma_var * 4;

printf("Resultat = %d\n", ma_var);

}
```

```c
int main()
{

int ma_var = 4;
ma_var *= 4;

printf("Resultat = %d\n", ma_var);

}
```

- Exemples :
  - variable += nombre, variable -= nombre, variable %= nombre, variable /= nombre

Opérations mathématiques

- le type d'une opération (le type du résultat d'une opération), dépendait de celui de ses opérandes. Or, si c'est évident dans le cas où les opérandes ont le même type, cela ne l'est pas dans le cas où leurs types sont différents!
- dans le cadre d'opérations, des conversions peuvent avoir lieu d'un type à l'autre. Ces conversions sont appelées des **conversions implicites**, car elles sont effectuées automatiquement.
- L'expression d'affectation peut provoquer une conversion de type. Exemple:

```
int i;
float x;
```

- Si i vaut 3, l'expression x = i donne à x la valeur 3.0 (conversion entier → réel).
- Si x vaut 4.21, l'expression i = x donne à i la valeur 4, partie entière de x (conversion réel → entier).
- On peut également provoquer une conversion de type grâce à l'opérateur **() (type casting).**
- **Exemple:**

  (int)x est de type **int** et sa valeur est la partie entière de x.

Opérations mathématiques

Exemple de conversion de type:

```
int main()
{
    int a = 5;
    int b = 2;

    printf("%f\n", a / (double)b);

    printf("%f\n", (double)a / (double)b);
}
```