INFORMATIQUE 1

V. PSEUDO-CODE, C ET VARIABLES

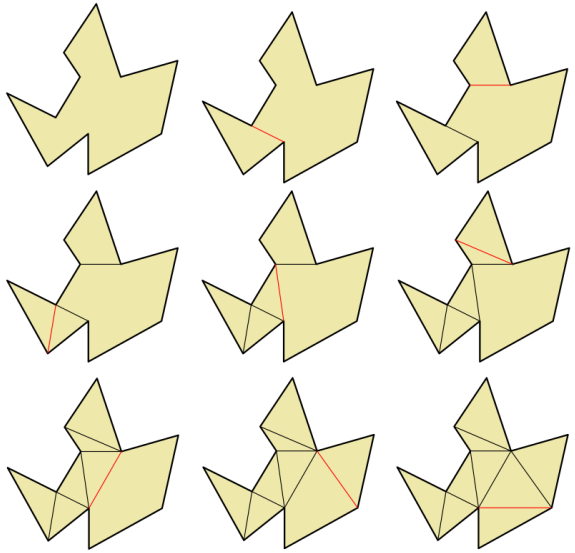
Eva Ansermin & Romuald Grignon

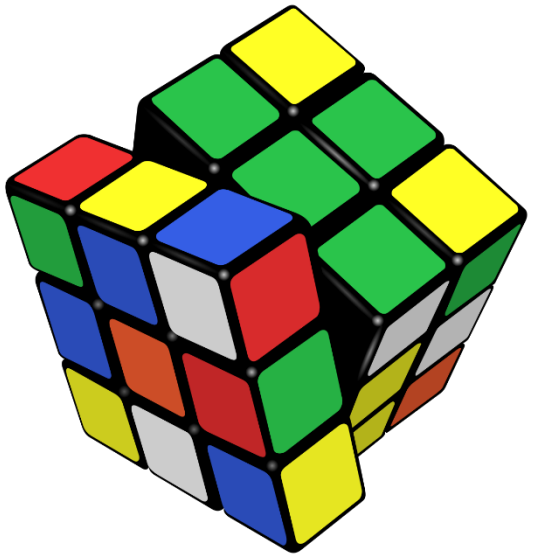
CY TECH

I. Introduction : l'algorithmie

Algorithmie vs programmation

- L'algorithmie :
  - Concevoir une solution pour résoudre un problème donné.
  - Un algorithme consiste en une succession d'étapes pour résoudre ce problème.
  - Ce problème n'est pas forcement mathématique ou informatique !





- Exemples : étapes de pliage ou résolution d'un rubik's cube

Algorithmie vs programmation

Problème: Gâteau aux pommes

Ingrédients

Nombre de gâteau - 1 +


120 g de sucre semoule


1 paquet de sucre vanillé


3 oeufs


0.5 paquet de levure chimique


125 g de farine


0.5 verre d'huile de colza


3 pommes

Préparation

TEMPS TOTAL : 57 MIN

Préparation : 12 min **Cuisson :** 45 min

Etape 1

Découper les pommes en petits cubes.

Etape 2

Préchauffer le four à 160°C (thermostat 5-6).

Etape 3

Mélanger tous les ingrédients de la pâte, puis ajouter les pommes préalablement coupées en petits cubes.

Etape 4

Mélanger à nouveau à la spatule.

Etape 5

Verser la pâte à gâteau dans un moule beurré et cuire pendant 40 à 50 minutes.

Etape 6

Déguster !

Evaluation de l'algorithme

- **Plusieurs solutions** peuvent exister pour répondre à **un même problème**. Il n'est pas toujours facile de trouver la plus efficace.

- Un programme peut chercher à optimiser :
  - Le temps d'exécution
  - La mémoire utilisée
  - Limiter sa consommation de ressources (fichiers ou électricité par exemple)
  - La sécurité / La prise en compte de tous les cas possibles

Evaluation de l'algorithme

- Des petites erreurs d'algorithmes peuvent impacter gravement de nombreux secteurs (financiers, sécurité etc.)
- Exemple du premier vol d'Ariane 5 (4 juin 1996) / Agence spatiale européenne :
  - Explosion de la fusée après 37 s de décollage
  - Une erreur de calcul du pilote automatique à cause d'une mauvaise indication des accéléromètres
  - La raison : la valeur mesurée par les accéléromètres ne tenait pas dans les emplacements mémoire de la fusée (1 octet) !

- Progrès de l'algorithmie!

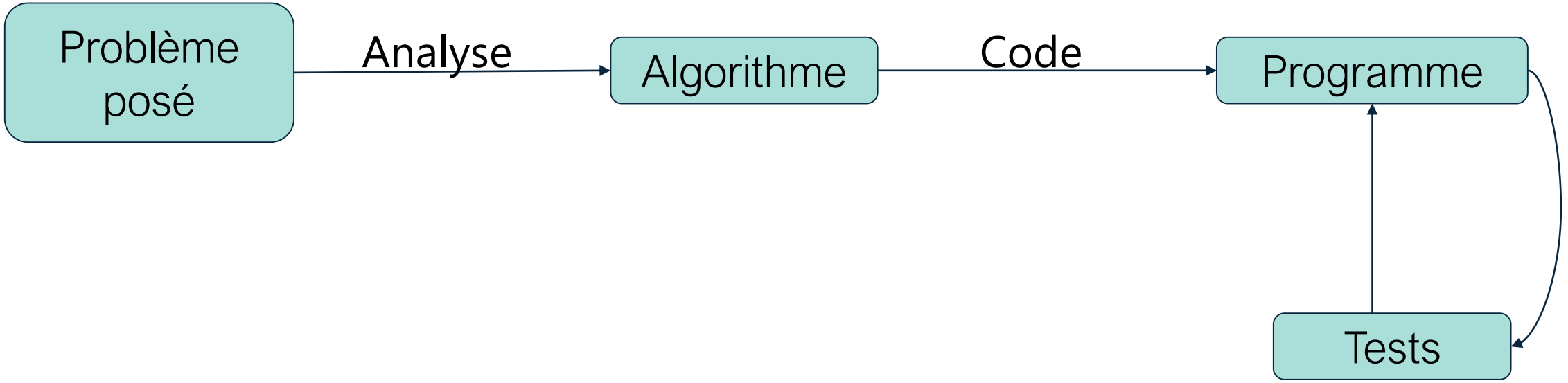
Algorithmie vs programmation

- L'algorithmie :
  - Conception de solution pour résoudre un problème donné.
  - Un algorithme est une succession d'étapes pour résoudre ce problème.

- La programmation :
  - Dire à un ordinateur ce qu'il doit faire.
  - Ecrire les instructions dans un langage compréhensible par la machine (C, python, pascal…)
  - On utilise les concepts de l'algorithmie pour concevoir les programmes.

➢ Un programme est issu d'un algorithme !

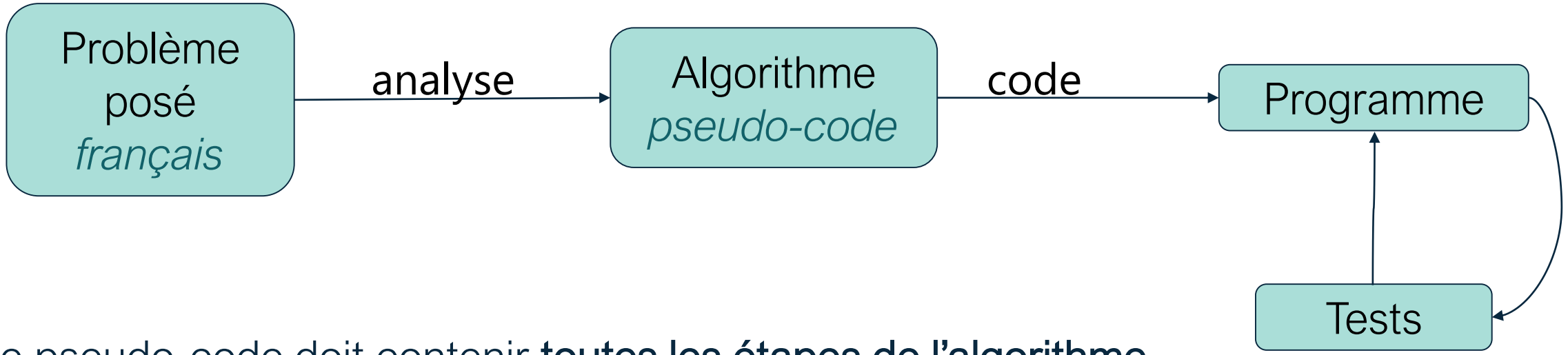
Etapes de conception d'un programme



- Comment écrire un algorithme ?

Le pseudo-code

- En programmation le pseudo-code est une manière d'écrire un algorithme dans un **langage naturel**, indépendamment d'un langage de programmation. Il utilise notamment des mots clefs en français qui ont des équivalents dans les différents langages.



- Le pseudo-code doit contenir **toutes les étapes de l'algorithme**.
- **Il n'existe pas de convention unique** de pseudo-code : l'important est d'être compréhensible. Néanmoins, pour ce cours nous imposerons une convention.

II. Premiers programmes et variables

Programme

- Un **PROGRAMME** est l'unité de base d'un algorithme
- Un **PROGRAMME** possède un **DEBUT** et une **FIN**
- Syntaxe :

```
PROGRAMME NomDuProgramme
DEBUT

   ALGORITHME

FIN
```

Programme

- Un **PROGRAMME** est l'unité de base d'un algorithme
- Un **PROGRAMME** possède un **DEBUT** et une **FIN**
- Syntaxe :

```
PROGRAMME NomDuProgramme
DEBUT
   // Commentaire sur une ligne
   /*

   Commentaire
   sur plusieurs lignes
   */
FIN
```

Un commentaire est une note à destination du programmeur ou d'un futur lecteur.
Il ne fait pas partie de l'algorithme mais permet de l'expliquer.

Instructions

- Un **PROGRAMME** est une suite finie d'instructions, comprises entre le **DÉBUT** et la **FIN**
- Une instruction de base **ÉCRIRE** est l'écriture d'informations sur une interface

  ÉCRIRE("qqch à afficher")

L'instruction **ÉCRIRE** doit respecter une syntaxe !

- Exemple de programme complet : « Bonjour »

```
PROGRAMME Bonjour
DÉBUT
      ÉCRIRE("Bonjour le monde !")
FIN
```

Variables, types et expressions

- Variable : élément du programme qui sert à **stocker** une valeur, un résultat de calcul.
- Une variable est définie par un nom et un type.
  - Le nom d'une variable est donné par le programmeur : il permet simplement de pouvoir distinguer les différentes variables pour les utiliser.
  - Le type d'un variable indique quel genre de donnée elle représente (une lettre, un entier, un réel etc…), soit pour l'ordinateur sa taille en bits et sa convention de représentation !

Variables, types et expressions

- Variable : élément du programme qui sert à **stocker** une valeur, un résultat de calcul.
- Une variable est définie par un nom et un type.

- Les différents types en pseudo-code :
  - Arithmétiques
    - Entier : -7, 0, 42, 2147483647
    - Réel : -3.7, 0.0, 8.0, 1.42e-5
  - Logiques
    - Booléen : vrai, faux
  - Alphabétiques
    - Caractère : ‘a’, ‘Z’, ‘+’, ‘5’, ‘ ‘
    - Chaîne : “aaa”, “322 - 78”, “Bonjour Machin”

Les types en pseudo-code précisent simplement la « nature » de la variable et non la taille.

Variables, types et expressions

- On peut modifier la valeur des variables grâce à des opérateurs.
- L'application d'une suite d'opérations sur une ou plusieurs variables est une expression.
- Les opérateurs possibles dépendent du type :
  - Arithmétiques
    - Entier : -7, 0, 42, 2147483647, opérateurs : **+, -, *, DIV, MOD**
    - Réel : -3.7, 0.0, 8.0, 1.42e-5, opérateurs : **+, -, *, /**
  - Logiques
    - Booléen : vrai, faux, opérateurs : ou, et, non
  - Alphabétiques
    - Caractère : 'a', 'Z', '+', '5', ' '
    - Chaîne : "aaa", "322 - 78", "Bonjour Machin", opérateur : + (concaténation)

dividende	diviseur
reste	quotient

Ne pas confondre DIV et / !

Variables, types et expressions

- Les variables sont stockées dans la mémoire de l'ordinateur (la RAM).
- Il est important de **déclarer** une variable pour lui faire de la place en mémoire !
- Une variable est d'abord **déclarée** dans la zone de déclaration, puis **initialisée** dans la zone d'instructions.
- La zone de déclaration est définie par le mot clé `VARIABLE`
- Syntaxe : `nomVariable : type`

- Exemple de déclaration de variables

```
VARIABLES
  i             : entier
  x, y          : réels
  estTrouvé     : booléen
  reponse       : chaîne
```

Affectation

- Une **affectation** est le fait de mettre une **valeur** dans une **variable**, en cohérence avec son type.
- La première affectation de valeur à une variable est appelée **initialisation.**
- En pseudo-code, elle est symbolisée par ←
- ← peut être traduit par "*la variable prend la valeur*" :
  - `variable ← valeur`

Affectation

- Exemples

```
VARIABLES
   i, j          : entiers
   x, y          : réels
   estTrouvé     : booléen
   reponse       : chaîne
DEBUT
      i ← 7
      x ← 2.5
      j ← i + 1
      reponse ← “Bonjour “
      estTrouvé ← faux
      y ← 5 + 2 * x
 FIN
```

Affectation

- Exemples

```
VARIABLES
   i, j          : entiers
   x, y          : réels
   estTrouvé     : booléen
   reponse       : chaîne
DEBUT
      i ← 7
      x ← 2.5
      j ← i + 1
      reponse ← "Bonjour "
      estTrouvé ← faux
      y ← 5 + 2 * x
FIN
```

Zone de déclaration

Zone d'instructions

Affectation

- Exemples

```
VARIABLES
   i, j          : entiers
   x, y          : réels
   estTrouvé     : booléen
   reponse       : chaîne
DEBUT
      i ← 7
      x ← 2.5
      j ← i + 1
      reponse ← "Bonjour "
      estTrouvé ← faux
      y ← 5 + 2 * x
   FIN
```

première instruction

troisième instruction

dernière instruction

Les instructions sont exécutées dans l'ordre ligne par ligne !

Affectation

- Exemples

```
VARIABLES
   i, j          : entiers
   x, y          : réels
   estTrouvé     : booléen
   reponse       : chaîne
DEBUT
      i ← 7
      x ← 2.5
      j ← i + 1
      reponse ← “Bonjour “
      estTrouvé ← faux
      y ← “p“
      j ← 3.2
   FIN
```

Affectation

- Exemples

```
VARIABLES
   i, j            : entiers
   x, y            : réels
   estTrouvé       : booléen
   reponse         : chaîne
DEBUT
      i ← 7
      x ← 2.5
      j ← i + 1
      reponse ← “Bonjour “
      estTrouvé ← faux
      y ← “p“
      j ← 3.2
   FIN
```

Ces deux affectations sont fausses car elles ne sont pas compatibles avec les types des variables !

Exemple de programme

```
VARIABLES
   a, b,c  : entiers
   DEBUT
      a ← 2
      b ← 4
      c ← a * b
      a ← 1 + b div a
      b ← a-2*c
   FIN
```

Exemple de programme

```
VARIABLES
   a, b,c   : entiers
DEBUT
      a ← 2
      b ← 4
      c ← a * b
      a ← 1 + b div a
      b ← a-2*c
   FIN
```

variable	valeur
a	
b	
c	

Exemple de programme

```
VARIABLES
   a, b,c   : entiers
   DEBUT
      a ← 2
      b ← 4
      c ← a * b
      a ← 1 + b div a
      b ← a-2*c
   FIN
```

variable	valeur
a	2
b	
c	

Exemple de programme

```
VARIABLES
   a, b,c   : entiers
   DEBUT
      a ← 2
      b ← 4
      c ← a * b
      a ← 1 + b div a
      b ← a-2*c
   FIN
```

variable	valeur
a	2
b	4
c	

Exemple de programme

```
VARIABLES
   a, b,c   : entiers
   DEBUT
      a ← 2
      b ← 4
      c ← a * b
      a ← 1 + b div a
      b ← a-2*c
   FIN
```

variable	valeur
a	2
b	4
c	8

Exemple de programme

```
VARIABLES
   a, b,c   : entiers
   DEBUT
      a ← 2
      b ← 4
      c ← a * b
      a ← 1 + b div a
      b ← a-2*c
   FIN
```

variable	valeur
a	2 → 3
b	4
c	8

Exemple de programme

```
VARIABLES
   a, b,c   : entiers
   DEBUT
      a ← 2
      b ← 4
      c ← a * b
      a ← 1 + b div a
      b ← a-2*c
   FIN
```

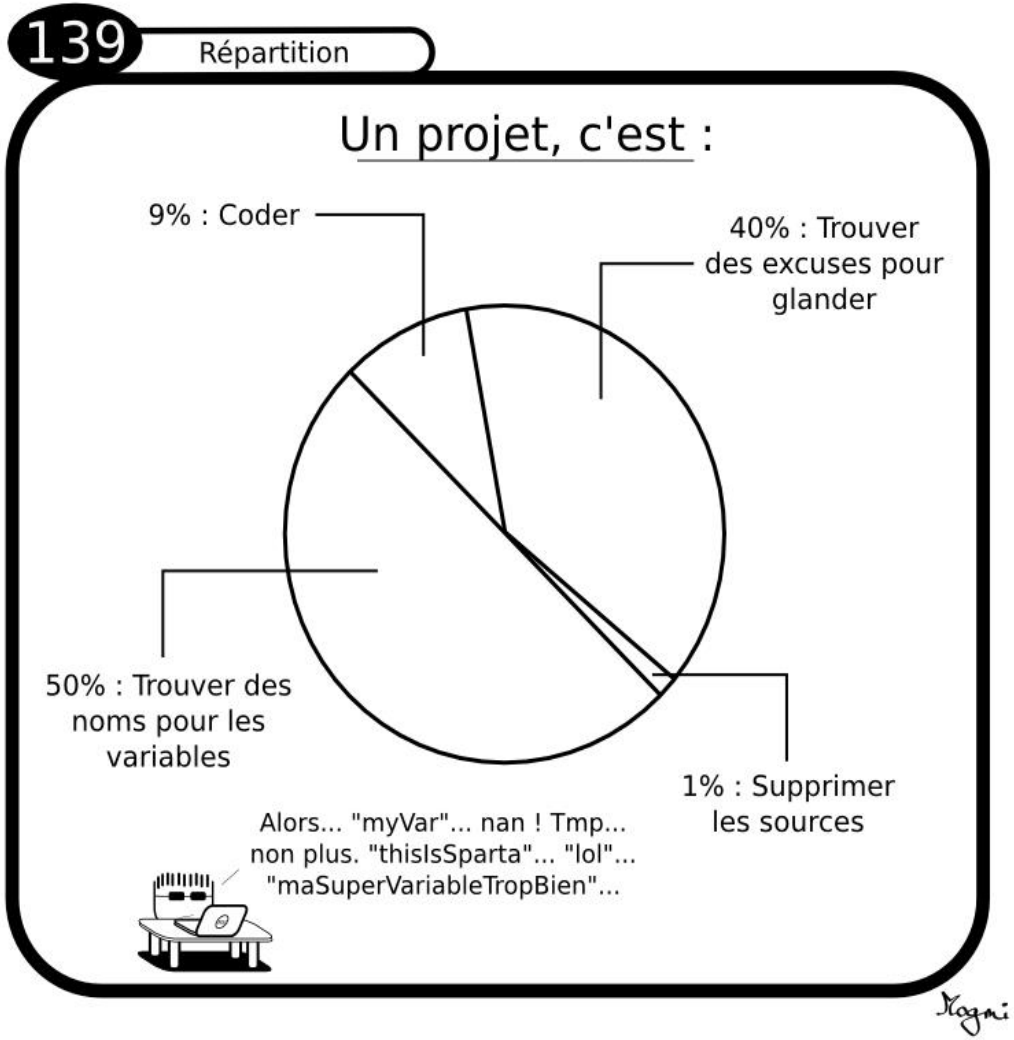
variable	valeur
a	2 → 3
b	4 → -13
c	8

Exemple de programme

```
VARIABLES
   a, b,c   : entiers
   DEBUT
      a ← 2
      b ← 4
      c ← a * b
      a ← 1 + b div a
      b ← a-2*c
➡  FIN
```

variable	valeur
a	2 → 3
b	4 → -13
c	8

Attention aux noms des variables!



Interactions avec l'utilisateur

- On modélise deux interactions : **l'entrée** des données (la saisie) et **la sortie** (l'affichage).
- Sortie : **`écrire(`*`expression`*`)`**
- Exemple :
  - Texte simple : `écrire("Vive les chats!")`
  - Texte + valeur d'une variable : `écrire("ma variable a vaut " + a)`
- Entrée : **`lire(`*`variable`*`)`**
- La *lecture* correspond à l'affectation de la valeur saisie (au clavier) dans la variable spécifiée
- Exemple : **`lire(`*`a`*`)`**

Un exemple complet

```
PROGRAMME Interaction
VARIABLE
      reponse : chaîne
DÉBUT
      // affichage
      écrire(“Saisissez votre prénom”)
      // saisie clavier
      lire(reponse)
      écrire(“Bonjour “ + reponse + “ !”)
FIN
```

Ecran:

Un exemple complet

```
PROGRAMME CalculSimple
VARIABLE
      nb : ENTIER
DÉBUT
      // affichage
      écrire(“Saisir un nombre entier“)
      lire(nb) //nb prend la valeur saisie au clavier
      // on incrémente
      nb ← nb + 1
      écrire(“Voici le suivant : “ + nb)
FIN
```

Ecran:

Un exemple complet

variable	valeur
nb	

```
PROGRAMME CalculSimple
VARIABLE
      nb : ENTIER
DÉBUT
      // affichage
      écrire("Saisir un nombre entier")
      lire(nb) //nb prend la valeur saisie au clavier
      // on incrémente
      nb ← nb + 1
      écrire("Voici le suivant : " + nb)
FIN
```

Ecran:

Un exemple complet

variable	valeur
nb	

```
PROGRAMME CalculSimple
VARIABLE
      nb : ENTIER
DÉBUT
      // affichage
      écrire("Saisir un nombre entier")
      lire(nb) //nb prend la valeur saisie au clavier
      // on incrémente
      nb ← nb + 1
      écrire("Voici le suivant : " + nb)
FIN
```

Ecran:
Saisir un nombre entier

Un exemple complet

variable	valeur
nb	5

```
PROGRAMME CalculSimple
VARIABLE
      nb : ENTIER
DÉBUT
      // affichage
      écrire("Saisir un nombre entier")
      lire(nb) //nb prend la valeur saisie au clavier
      // on incrémente
      nb ← nb + 1
      écrire("Voici le suivant : " + nb)
FIN
```

Ecran:
Saisir un nombre entier

Un exemple complet

variable	valeur
nb	6

```
PROGRAMME CalculSimple
VARIABLE
      nb : ENTIER
DÉBUT
      // affichage
      écrire("Saisir un nombre entier")
      lire(nb) //nb prend la valeur saisie au clavier
      // on incrémente
      nb ← nb + 1
      écrire("Voici le suivant : " + nb)
FIN
```

Ecran:
Saisir un nombre entier

Un exemple complet

variable	valeur
nb	6

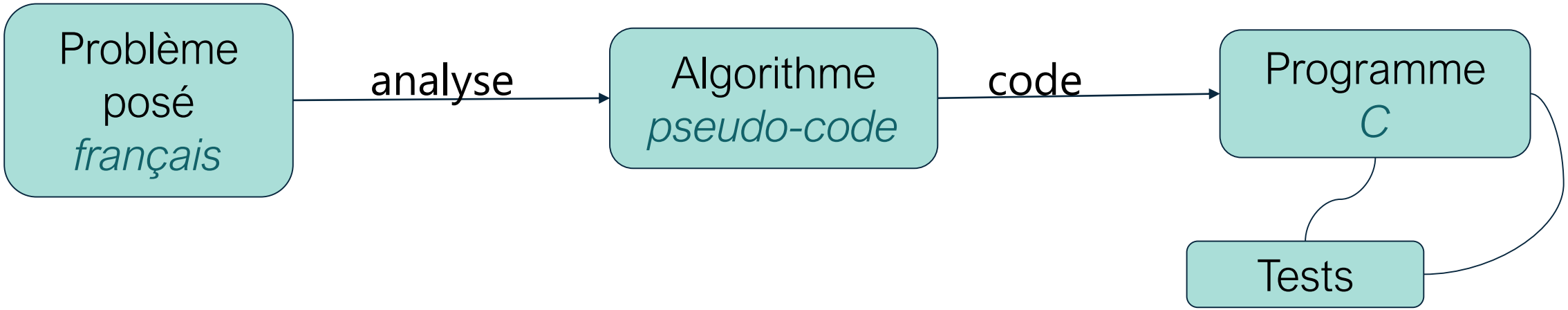
```
PROGRAMME CalculSimple
VARIABLE
      nb : ENTIER
DÉBUT
      // affichage
      écrire("Saisir un nombre entier")
      lire(nb) //nb prend la valeur saisie au clavier
      // on incrémente
      nb ← nb + 1
      écrire("Voici le suivant : " + nb)
FIN
```

Ecran:
Saisir un nombre entier
Voici le suivant : 6

III. Le langage C

La programmation sur ordinateur

- Une fois l'algorithme écrit, il faut le traduire dans un langage compréhensible par la machine



- Il existe de très nombreux langages de programmation, chacun ayant des spécificités qui les rendent plus ou moins efficaces pour une application particulière.

La programmation sur ordinateur

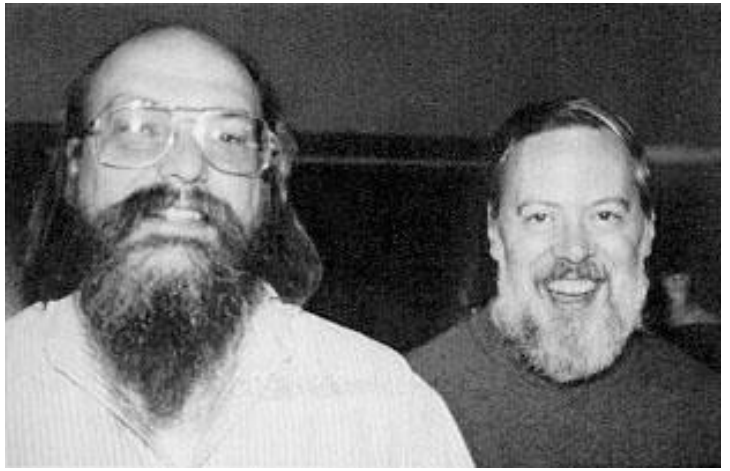
Plusieurs approches de programmation :

- **fonctionnelle** : appels de fonctions mathématiques (Caml, Scheme, Haskell) ;
- **procédurale** : suite d'instructions ou d'appels de fonctions ⇒ automate (C, Pascal, Fortan) ;
- **logique** : utilisation de formules logiques (Prolog) ;
- **objet** : ensemble d'objets communiquant ensemble (Java, C++, C#).

On distingue deux types de langages selon leur proximité avec la machine:

- Bas niveau
- Haut niveau

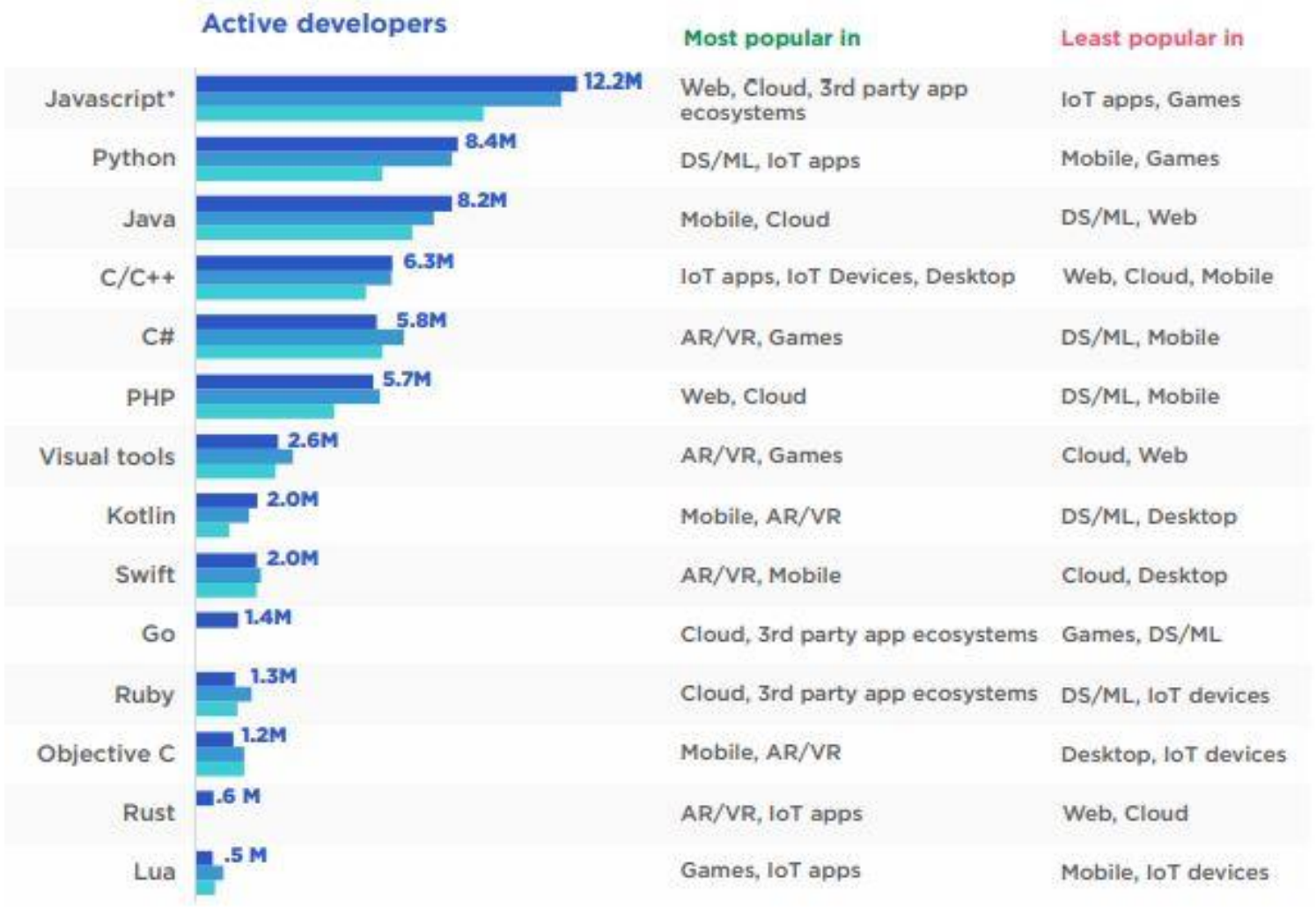
Le langage C



- Créé en 1972 par Ken Thompson & Dennis Ritchie ;
- Compromis entre le langage de bas niveau (assembleur) et de haut niveau (Fortran) ;
- Rapide : pas de vérification de gestion de la mémoire, elle est à la charge du développeur !
- Moyennement typé : chaque variable doit être typée, mais certaines opérations entre différents types sont possibles.

Le langage C

Active software developers, globally, in millions Q4 2019 (n=12,066)



/DATA

Le langage C

Nov 2021	Nov 2020	Change	Programming Language	Ratings	Change
1	2	↑	Python	11.77%	-0.35%
2	1	↓	C	10.72%	-5.49%
3	3		Java	10.72%	-0.96%
4	4		C++	8.28%	+0.69%
5	5		C#	6.06%	+1.39%
6	6		Visual Basic	5.72%	+1.72%
7	7		JavaScript	2.66%	+0.63%
8	16	⇈	Assembly language	2.52%	+1.35%
9	10	↑	SQL	2.11%	+0.58%
10	8	↓	PHP	1.81%	+0.02%

Index tiobe

Le langage C

- Base de très nombreux langages : facile d'apprendre un autre langage si on connait le C.
- Permet de comprendre le fonctionnement de l'ordinateur.
- Optimisation de la mémoire utilisée et rapidité.

- Syntaxe stricte.
- Certains concepts difficiles (pointeurs, listes chaînées etc...).
- Un peu de temps avant de pouvoir produire des choses sympathiques…

Un exemple

```c
/**
 * Mon premier programme
 */
#include <stdio.h>

int main(){

    printf("Hello World");
    return 0;
}
```

```
/**
 * Mon premier programme
 */

DEBUT

      ECRIRE(“Hello World”)
FIN
```

Un exemple

```c
/**
 * Mon premier programme
 */
#include <stdio.h>

int main(){

    printf("Hello World");
    return 0;
}
```

```
/**
 * Mon premier programme
 */

DEBUT

      ECRIRE(“Hello World”)
FIN
```

Toute instruction se termine par un ;

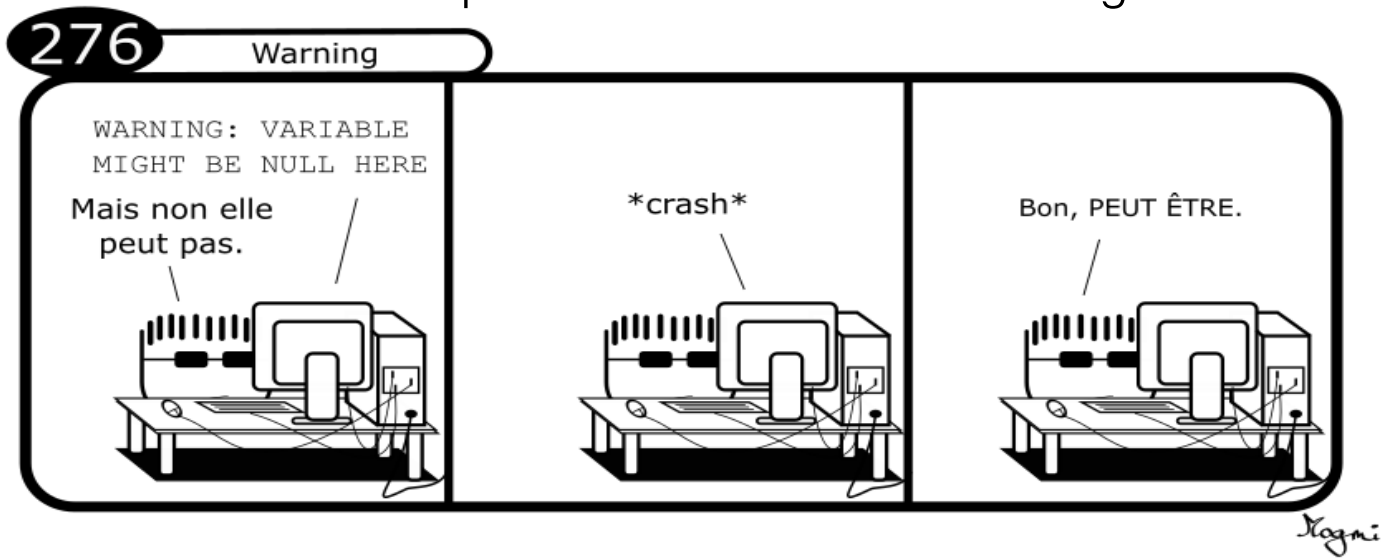
La compilation

- Le code source, compréhensible par un humain (ou presque) est enregistré dans un fichier ***.c**
- L' ordinateur ne «comprend » pas directement le C : il ne comprend que le binaire!
- Un logiciel appelé compilateur permet de traduire le programme en un fichier binaire, exécutable par l'ordinateur.

- Le C est un langage compilé (un compilateur créé un exécutable). Il existe au contraire des langages interprétés (la traduction en langage machine se déroule en même temps que l'exécution).

La compilation

- Lorsqu'il y a des problèmes dans le code (syntaxe, problèmes de types etc…), le compilateur peut faire apparaitre des messages :
  - Des erreurs : la compilation ne peut pas se faire. Il s'agit souvent d'erreurs de syntaxe (absence de point-virgule, parenthèse ou accolade non fermée etc.)
  - Des avertissements : la compilation peut se faire mais il est possible qu'il y ait un problème à l'exécution. Il s'agit souvent d'erreurs de type.



La compilation

- Lorsqu'il y a des problèmes dans le code (syntaxe, problèmes de types etc…), le compilateur peut faire apparaitre des messages :
  - Des erreurs : la compilation ne peut pas se faire. Il s'agit souvent d'erreurs de syntaxe (absence de point-virgule, parenthèse ou accolade non fermée etc.)
  - Des avertissements : la compilation peut se faire mais il est possible qu'il y ait un problème à l'exécution. Il s'agit souvent d'erreurs de type.

- Ce n'est pas parce qu'un code compile qu'il fonctionne correctement ou qu'il fait ce que vous avez prévu !

La compilation

- Sous Linux, le **compilateur gcc** s'occupe de traduire le langage C en langage machine. Il produit un fichier exécutable :
  - utilisation : **gcc -o monExecutable monCode.c**
  - exécution : **./monExecutable**

- Options
  - **-Wall : Affiche tous les warnings par le compilateur**
  - -Werror : Transforme tous les warnings en erreurs
  - -o nomFichierExecutable : Précise le nom du fichier en sortie
  - Pour plus d'options, voir le manuel de gcc

Les variables

```c
int main() {
    // déclaration
    int a, b;
    float d;

    // affectation
    a = 12;
    b = a;
    d = 3.14;

    return 0;
}
```

```
VARIABLE
     a, b : ENTIER
     d : REEL
DEBUT

    // affectation
    a <- 12
    b <- a
    d <- 3,14

FIN
```

Les variables

- Déclaration : nécessite un nom et un type : **`type nom;`**
  - `int a;`
  - `float x;`
- affectation : =
  - a = 1;
- Attention au nommage de variables !
  - Commence par une **lettre minuscule**
  - **Pas de nom sur plusieurs mots (pas d'espace)**

```
int main() {
    // déclaration
    int a, b;
    float d;

    // affectation
    a = 12;
    b = a;
    d = 3.14;

    return 0;
}
```

- Il faut déclarer les variables avant de les utiliser !

Les variables

- Types de base :
  - char : caractères ASCII codé sur 1 octet,
  - int : -2147483648 à 2147483647 codé sur 4 octets,
  - float : (+-) : $3.4 * 10^{-38}$ à $3.4 * 10^{38}$ codé sur 4 octets ;
- **Pas de booléen en C** (on gère avec des entiers (0 : faux, autre pour vrai)) ;
- Pas de type *chaîne de caractères* (on verra ça plus tard!)

Type (pseudo-code)	Type en C	Encodage dans ordinateur	Taille (pour linux)
Entier signé	int	CC2	32 bits
Caractère	char	Binaire	8 bits
Réel	float	Virgule flottante	32 bits
“gros” entier	long	CC2	64 bits
“gros” reel	double	Virgule flottante	64 bits
Entier positif	unsigned int	Binaire	32 bits
…	…	…	…

Attention aux types ! Un exemple historique….

- Avant Gnagnam-style, le compte de like sur youtube **était encodé en int sur 32 bit.** La valeur maximum possible de likes était alors de 2,147,483,647
- La vidéo était bloquée sur ce nombre de vues, mais si on ajoutait un like on pouvait observer un joli nombre négatif de pouces bleus…
- Maintenant c'est 64 bits… soit neuf milliards de milliards (9 * 10^18)

Opérations

- Numériques +, -, * , Division : / (entiers ⇔ DIV)
- Modulo : % (entiers ⇔ MOD)
- Quelque raccourcis:
  - **a++; ⇔ a=a+1;**
  - a--; ⇔ a=a-1
  - a+=5; ⇔ a=a+5;
  - a-=5; ⇔ a=a-5;
  - a*=y ⇔ a=a*y etc..

```
int main() {
    // déclaration
    int a, b = 0;
    float d;

    // affectation
    a++;            //a=1
    b-=a;           //b=-1
    d=a/(2*b);      //d=0 !!
    a+=2;           //a=3

    return 0;
}
```

Exemple

```
int main() {
    // déclaration
    int a, b = 0;
    float d;

    // affectation
    a++;            //a=1
    b-=a;           //b=-1
    d=a/(2*b);      //d=0 car la division se fait entre
                      deux entiers !!
    a+=2;           //a=3
    return 0;
}
```

Entrée / **Sortie**

```c
#include <stdio.h>
int main() {
      // déclaration
    int a,res;
    // affectation
    a = 4;
    res=a*a;
    printf("le carré de a
            est %d",res);
    return 0;
}
```

```
VARIABLE
     a, res : ENTIER
DEBUT
    // affectation
    a <- 4
    res <- a*a
    ECRIRE("le carré de a
              est"+res)
FIN
```

Entrée / **Sortie**

```c
#include <stdio.h>
int main() {
      // déclaration
    int a,res;
    // affectation
    a = 4;
    res=a*a;
    printf("le carré de %d est %d", a, res);
    return 0;
}
```

Entrée / **Sortie**

- Les sorties en C sont formatées : on sépare les valeurs et la chaîne affichée (la forme et le fond)
  - `printf("coucou");`
  - `printf("carre(%d) = %d", x, x*x);`
- Formats :

Type (pseudo-code)	Type en C	Format associé
Entier	int	%d
Charactère	char	%c
Réel	float	%f
"gros" entier	long	%ld
"gros" reel	double	%lf
…	….	…

Entrée / Sortie

- Les entrées en C sont également formatées : on annonce ce que l'on va lire :
  - `scanf("%d", &monEntier);`
  - `scanf("%f", &monReel);`

Type (pseudo-code)	Type en C	Format associé
Entier	int	%d
Charactère	char	%c
Réel	float	%f
"gros" entier	long	%ld
"gros" reel	double	%lf
…	….	…

Entrée / Sortie

- Les entrées en C sont également formatées : on annonce ce que l'on va lire.
  - `scanf("%d", &monEntier);`

- Il sera interdit (par choix arbitraire, et non par limitation technique) :
  - de lire plus d'une variable par instruction scanf
    ~~`scanf("%d %d", &a, &b);`~~
  - de mettre du texte dans les "%..." du scanf ;
    ~~`scanf("saisir un nombre %d", &a);`~~

Les bibliothèques : stdio.h ?

- Les commandes **`printf`** et **`scanf`** sont des fonctions : elles font appel à du code déjà écrit.
- Une fonction peut être :
  - Ecrite par vous (voir dans 3 cours !)
  - Ecrite préalablement par d'autre programmeurs et stockée dans une **bibliothèque.**

- **`printf`** et **`scanf`** appartiennent à la bibliothèque **stdio.h**
- **`#include`** en début de code permet d'indiquer au compilateur que l'on va utiliser cette bibliothèque.

```
#include <nom_bibliotheque.h>
```

indique que l'instruction est réalisée avant la compilation (on verra plus tard !)

L'extension des bibliothèques est toujours .h !

En résumé

- Structure générale d'un programme C

```
#Instructions du préprocesseur
(inclusion bibliothèque)
```

```
int main(){
        déclarations des variables
        instructions
        return 0;
}
```

En résumé

- C’est normal d’avoir des erreurs à la compilation : le principal est d’arriver à les résoudre !

- Pour apprendre à coder, il faut coder !