UNIVERSITÉ
CÔTE D'AZUR

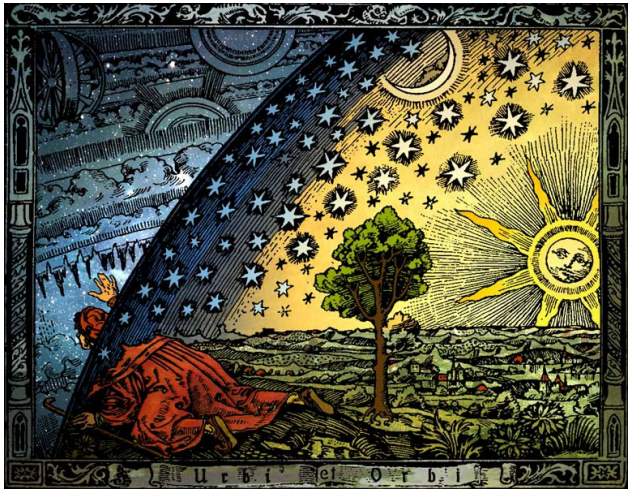
Langage C: Introduction

Présentation: **Stéphane Lavirotte**
Auteurs: ... et al*

(*) Cours réalisé grâce aux documents de :
Erick Gallesio, Stéphane Lavirotte

Mail: Stephane.Lavirotte@univ-cotedazur.fr
Web: http://stephane.lavirotte.com/
Université Côte d'Azur

UNIVERSITÉ
CÔTE D'AZUR



Les Paradigmes de Programmation

Langage C : Programmation Procédurale

UNIVERSITÉ CÔTE D'AZUR

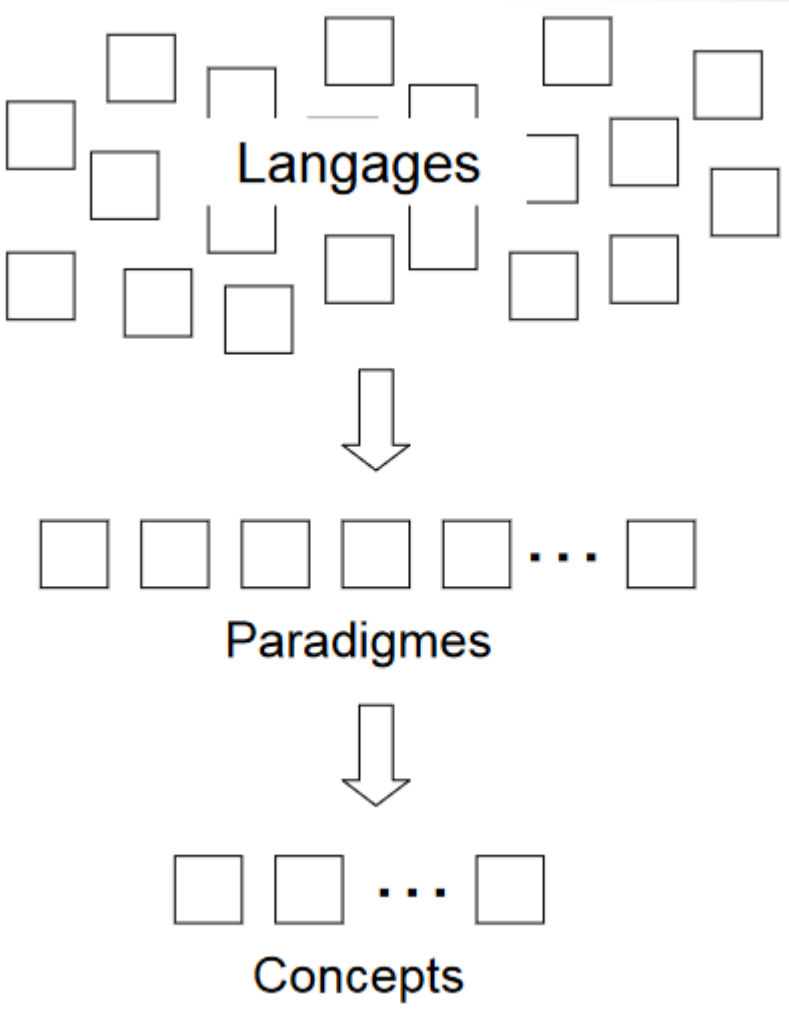
Paradigme : Définitions

- ✓ **Etymologie**
  - Le mot « paradigme » tient son origine du mot παράδειγμα (*paradeïgma*) en grec ancien qui signifie « modèle » ou « exemple »
  - Conception théorique dominante **ayant cours à une certaine époque** dans une communauté scientifique donnée, qui fonde les types d'explication envisageables, et les types de faits à découvrir dans une science donnée.
- ✓ **Dictionnaire Larousse**
  - En Logique: **Modèle théorique** de pensée qui oriente la recherche et la réflexion scientifiques.
- ✓ **Wikipedia**
  - Un paradigme est une **représentation du monde**, une **manière de voir les choses**, **un modèle cohérent du monde** qui réponse sur un fondement défini (matrice disciplinaire, modèle théorique, courant de pensée)
- ✓ **En Informatique**
  - Exprimer la façon dont un système a été conçu et pensé dans ses grandes lignes
  - Les **révolutions informatiques** coïncident généralement avec un **changement de paradigme**, où une vision différente des problèmes et de leur solution permet d'apporter une solution élégante techniquement et/ou ergonomiquement, à condition que l'utilisateur ou l'informaticien bascule vers le nouveau mode de réflexion exigé.

UNIVERSITÉ CÔTE D'AZUR

Paradigmes de Programmation

- ✓ Manière de programmer un ordinateur
  - Basé sur un ensemble de principes ou de théories
- ✓ Paradigmes et Concepts
  - Beaucoup plus de langages que de paradigmes
  - Beaucoup plus de paradigmes que de concepts
  - Un paradigme = un ensemble de concepts
  - Avec n concepts, on peut construire 2^n paradigmes (en théorie)



UNIVERSITÉ
CÔTE D'AZUR

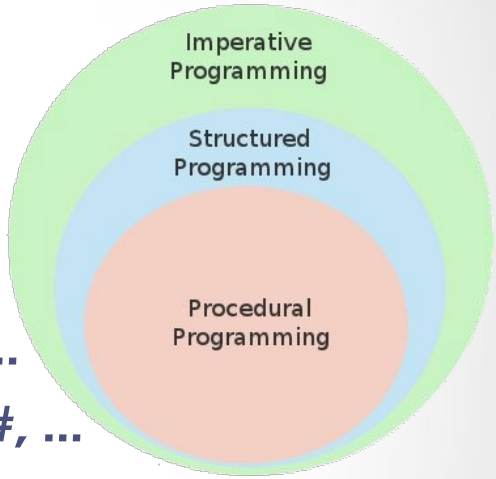
Les Concepts Majeurs

- ✓ L'enregistrement
- ✓ La fermeture
  - La continuation
  - L'exception
- ✓ L'indépendance
  - La concurrence
  - L'interaction: affectation unique, choix non-déterministe ou état partagé
- ✓ L'état
  - La variable affectable (1ère forme de l'état)
  - Le canal de communication (2ème forme de l'état)
- ✓ L'abstraction de données
  - L'encapsulation
  - Le polymorphisme
  - L'héritage
- ✓ La programmation orientée but
  - La programmation paresseuse
  - La programmation par contraintes

Source: Peter Van Roy, conférence UPMC

UNIVERSITÉ CÔTE D'AZUR

Familles de Programmation



- ✓ Programmation Impérative
  - Programmation Structurée:
    - Éviter la programmation spaghetti (ex. goto)
  - Programmation Procédurale: C, Ada, Pascal, ...
  - Programmation Orientée Objet: C++, Java, C#, ...

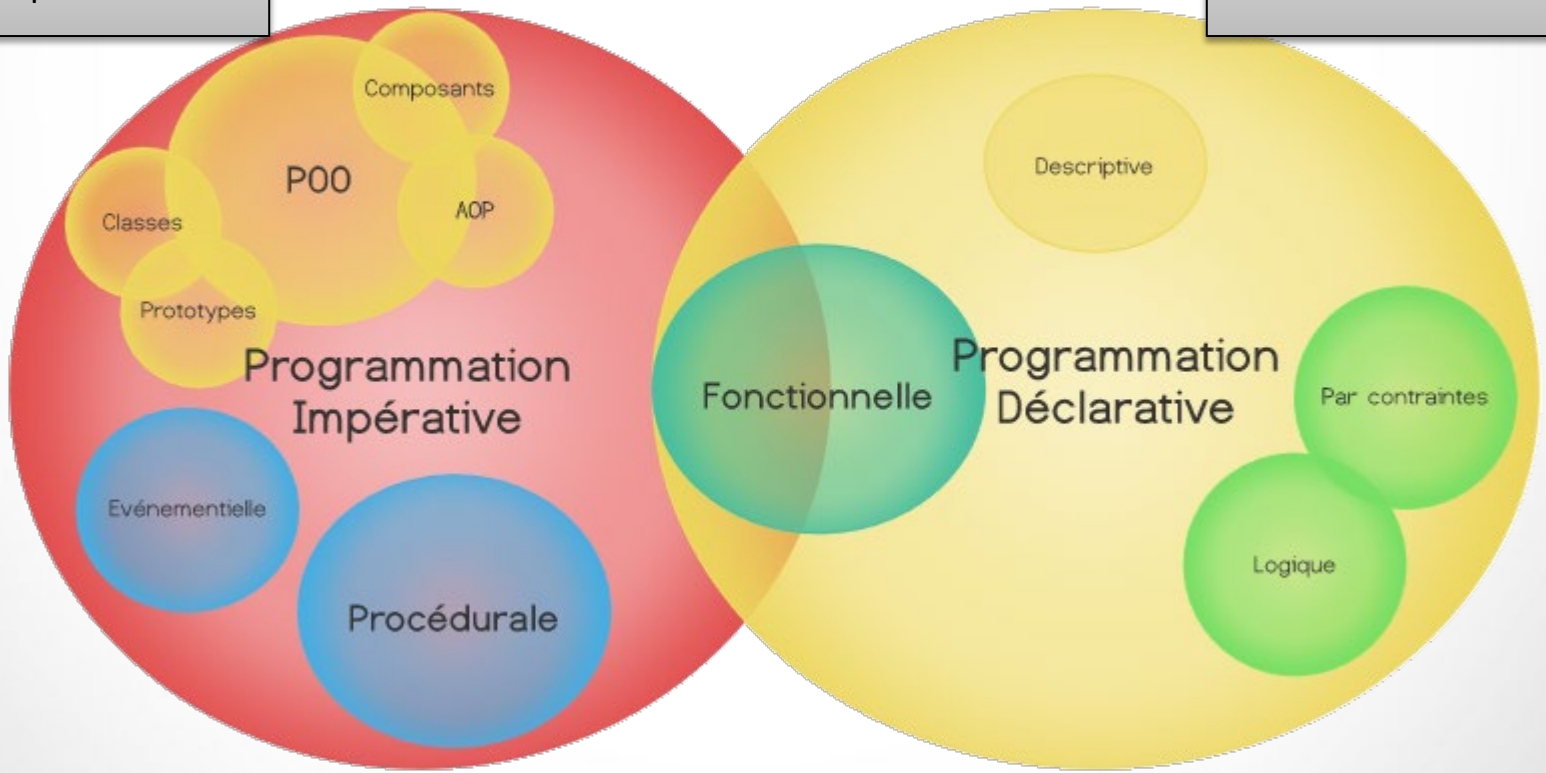
- ✓ Programmation Déclarative
  - Programmation Fonctionnelle
  - Programmation Logique
  - Programmation par Contraintes

UNIVERSITÉ CÔTE D'AZUR

Programmation Impérative vs Déclarative

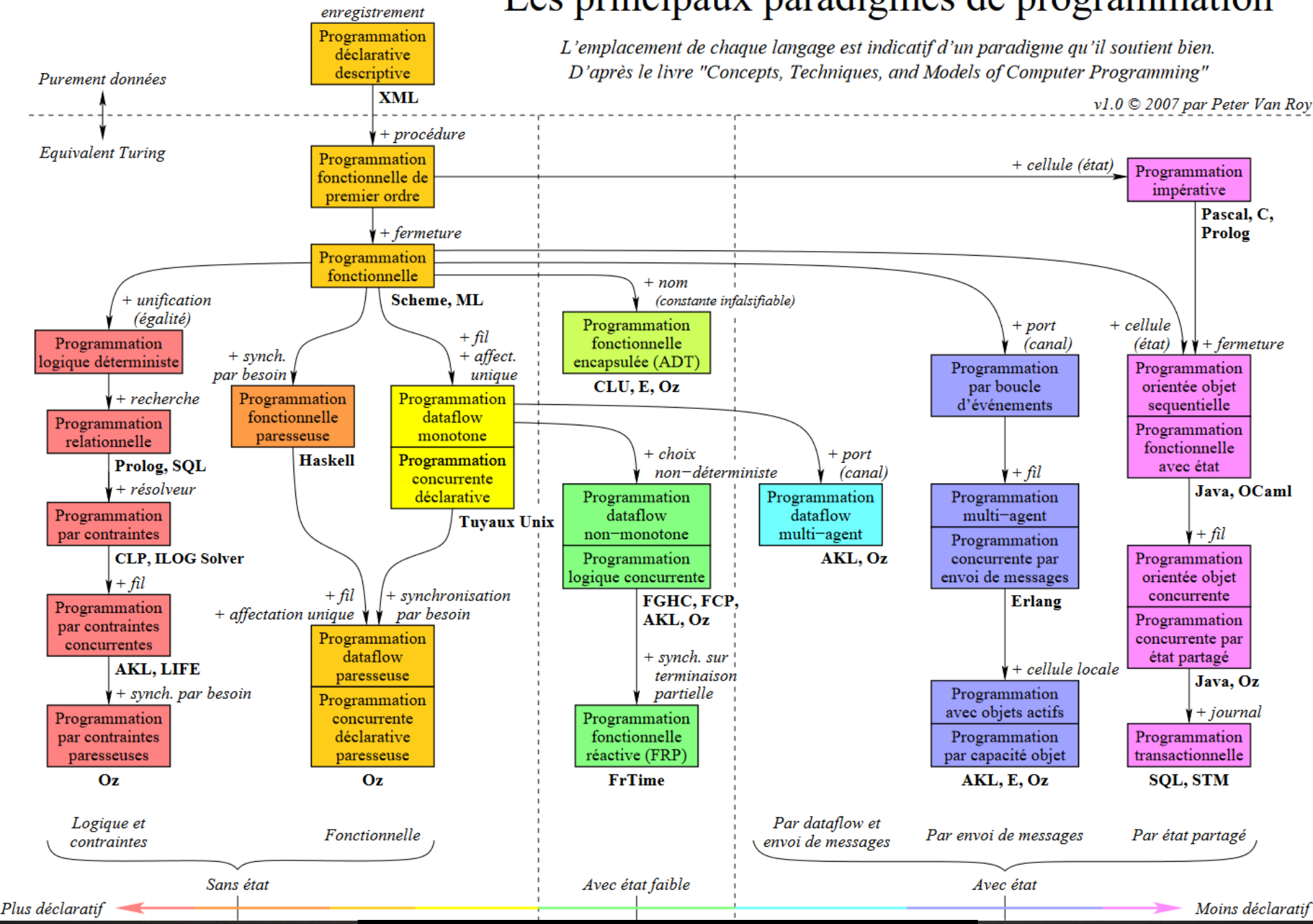
Comment ?
Comment le programme doit accomplir la tâche

Quoi ?
Ce que le programme doit atteindre le résultat



Les principaux paradigmes de programmation

L'emplacement de chaque langage est indicatif d'un paradigme qu'il soutient bien.
D'après le livre "Concepts, Techniques, and Models of Computer Programming"



UNIVERSITÉ CÔTE D'AZUR

Pourquoi apprendre le langage C ?

- ✓ A la base de nombreux langages (syntaxiquement)
  - C++, C#, Java, PHP, Javascript, ...
- ✓ A la base, le langage pour coder Unix
  - Noyau, compilateur, outils système, bibliothèques, ...
- ✓ Linux Torvalds: « Nothing better than C » (2012)
  - « *Si vous pensez comme un ordinateur, écrire en C a du sens* »
- ✓ Un des langages les plus efficaces
  - Efficacité au niveau du CPU
  - Consommation mémoire
- ✓ Un des langages les plus utilisés encore de nos jours
- ✓ Des centaines de millions de lignes écrites en C
  - Noyau Linux : ~30 millions de lignes de code; 98,5% en C
  - Compilateur gcc: ~10 millions de lignes de code; 47,7% en C

UNIVERSITÉ
CÔTE D'AZUR

Le Langage C

2022: le langage C a 50 ans

UNIVERSITÉ
CÔTE D'AZUR

Un peu d'histoire

- ✓ Racines
  - Algol 60
  - BCPL (1967)
  - B (auteur: K. Thomson ~ 1970)
- ✓ Auteur
  - Denis Ritchie (Bell Labs 1972)
  - Première spécification du langage dans le livre de Kernigham et Ritchie en 1978 (version K&R)
- ✓ Plusieurs versions de C disponibles:
  - Traditionnel: celui qui est décrit dans la version originale de K&R 1978 - Version obsolète
  - ANSI C et ISO C: sur-ensembles de K&R C (1983 → 1990)
  - Version la plus courante
    - C99 (1999) - La version que nous utiliserons dans ce cours
    - C11 (08/12/2011) - La dernière incarnation du langage (peu diffusée)

UNIVERSITÉ
CÔTE D'AZUR

Buts du langage 1/2

- ✓ Langage de programmation:
  - facile à apprendre
  - facile à implémenter
- ✓ Langage de programmation pour implémenter des systèmes d'exploitation:
  - conçu à l'origine pour programmer le noyau d'Unix v6
  - proche de la sémantique du processeur
  - simple et efficace
  - accès facile aux mécanismes de bas niveau
  - pointeurs (typés, mais non contrôlés)
  - contrôles de types permissifs (outil **lint**)
  - pas de contrôle de type à l'exécution

UNIVERSITÉ CÔTE D'AZUR

Buts du langage 2/2

- ✓ Permettre l'écriture de programmes portables
  - PCC (Portable C Compiler)
  - utilisation d'une bibliothèque standard
  - les E/S ne sont pas dans le langage
  - pas de chaînes de caractères à proprement parler (mais de nombreuses fonctions pour les manipuler)
  - pas de concurrence dans le langage (mais accessible au travers l'utilisation de fonctions de la bibliothèque)
- ✓ Fournir un ensemble d'outils système bien défini autour du langage
  - compilation séparée (on parle du système, ce qui permet d'éviter la définition d'extensions incompatibles)
  - options de compilation (on retrouve les même options sur la plupart des implémentations du langage)

UNIVERSITÉ CÔTE D'AZUR

Evaluation du langage 1/3

Avantages

- ✓ Efficace
  - registres, pointeurs, opérations sur les bits
  - pas de contrôle à l'exécution
  - . . .
- ✓ Grande liberté du programmeur (⇒ le programmeur est responsable)
- ✓ Bibliothèque très étendue
  - concurrence
  - E/S
  - manipulation de chaînes de caractères
  - . . .

⇒ des centaines de millions de lignes écrite en C

UNIVERSITÉ CÔTE D'AZUR

Evaluation du langage 2/3

Avantages (suite)

- ✓ Interface claire avec Unix
  - donne l'impression que l'on est sur Unix, même si ce n'est pas le cas.

  ⇒ permet l'écriture de programmes indépendants de l'OS cible (dans une certaine mesure)
- ✓ Bon langage d'assemblage portable utilisé comme langage cible par de nombreux compilateurs (C++, Objective C, Scheme, . . .)
- ✓ Vraiment portable
  - Unix en est la preuve (pratiquement entièrement écrit en C)
  - Linux qui tourne sur à peu près toutes les architectures matérielles (PC, Arm, Sun Sparc, PowerPC, 68000 machines, . . .)
  - Les outils du projet GNU de la FSF
  - compilateurs, éditeurs de texte, environnements de programmation, . . .

UNIVERSITÉ CÔTE D'AZUR

Evaluation du langage 2/3

Inconvénients

- ✓ Syntaxe à deux niveaux (préprocesseur)
- ✓ Grande liberté du programmeur (⇒ le programmeur doit être responsable)
- ✓ Les erreurs de compilation n'aident pas toujours
  - contrôles de type souvent trop permissifs
  - lint (pour K&R)
  - c'est moins vrai en C ANSI
  - malheureusement compatibilité K&R peut être un problème
  - utilisation des pointeurs peut être "délicate"
- ✓ Langage ancien
  - pas de concepts modernes (généricité, objets, modules, . . .)
  - la modularité est très simpliste
    - basée sur les fichiers
    - pas adaptée pour les gros projets en équipes (quoi que, le noyau Linux...)

UNIVERSITÉ CÔTE D'AZUR

Implémentation du langage C

- ✓ DEC PDP-11 en 1975
- ✓ PCC (Portable C Compiler) en 1978
  - implémentation publique
  - la source de la plupart des compilateurs pré-ANSI
- ✓ Aujourd'hui: accessible sur la plupart des processeurs

UNIVERSITÉ
CÔTE D'AZUR

Langage C et norme 1/2

- ✓ Plusieurs normes:
  - norme ANSI X3J11
- ✓ Il y a aussi une norme ISO: ISO-9899
- ✓ Version C99
  - fonctions inline
  - nombres complexes
  - tableaux de longueur variable
  - ...

UNIVERSITÉ
CÔTE D'AZUR

Langage C et norme 2/2

Aujourd'hui

- ✓ Les compilateurs sont conformes à la norme ISO / C99
- ✓ Principales améliorations / K&R
  - Meilleure portabilité:
    - librairie standard normalisée
    - parties dépendantes de l'implémentation clairement identifiées
    - internationalisation et localisation
  - Sécurité améliorée
    - Contrôles de types plus stricts
    - **const** et **volatile**
  - Compatibilité (quasi) totale avec le C K&R
    - peut être un piège parfois

Quelques exemples

Quoi de mieux pour débuter que quelques exemples de code ?...

UNIVERSITÉ CÔTE D'AZUR

Exemple 1: Hello World

- ✓ **Le classique hello world**

```
#include <stdio.h>
main()
{
   printf("Hello, world!\n");
}
```

- ✓ **Compilation:**

```
$ gcc -o hello hello.c
```

- ✓ **Exécution:**

```
$ hello
Hello, world!
$
```

UNIVERSITÉ CÔTE D'AZUR

Exemple 2: Programme multi-fichiers 1/4

```
extern void say_hello(void);     /* Fichier main.c */
main() {
   say_hello();
}
```

```
#include <stdio.h>                /* Fichier hello.c */
void say_hello(void) {
   printf("Hello, world!\n");
}
```

- ✓ **Compilation:**

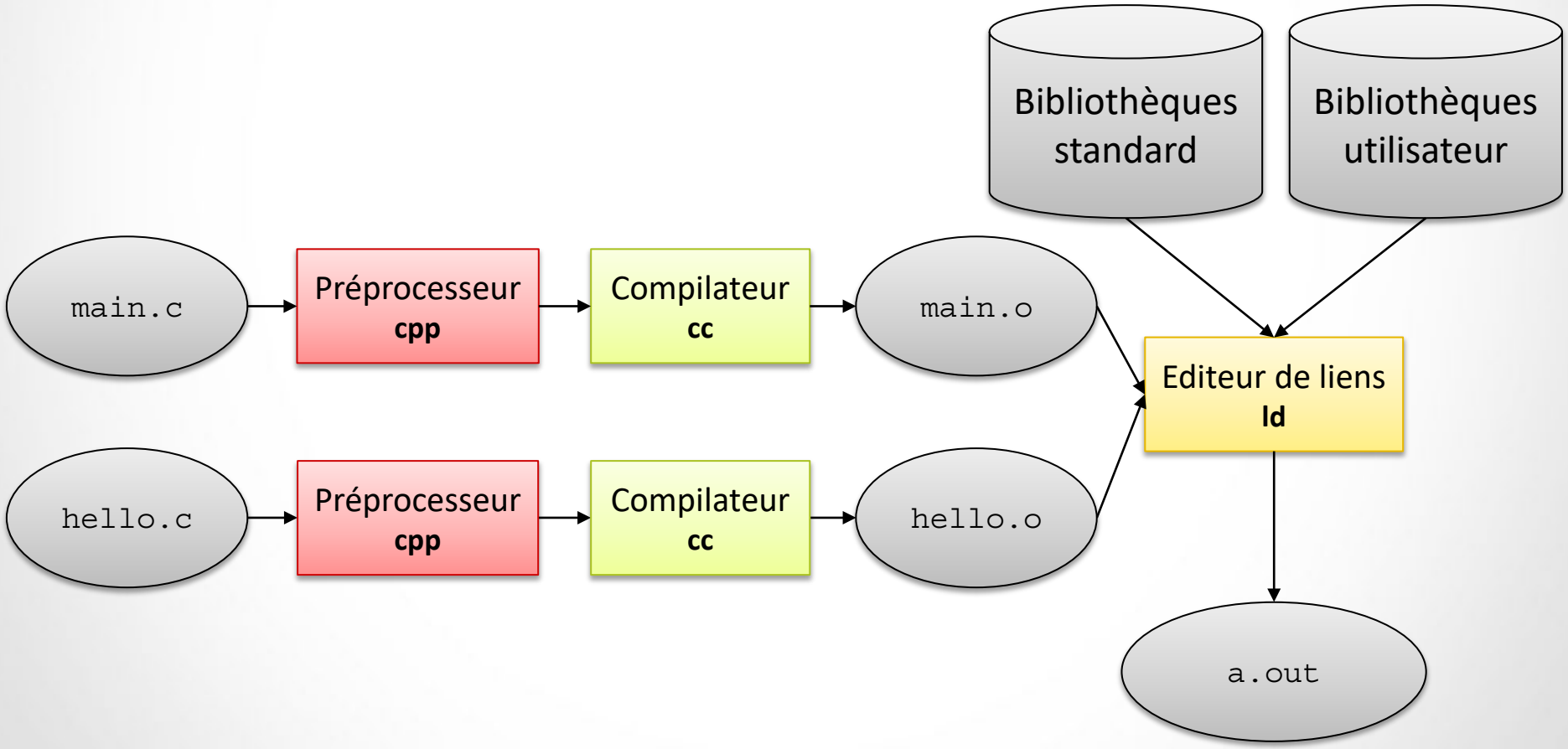
```
$ gcc -o hello main.c hello.c
```

- ✓ **ou encore**

```
$ gcc -c main.c
$ gcc -c hello.c
$ gcc -o hello main.o hello.o
```

UNIVERSITÉ CÔTE D'AZUR

Exemple 2: Programme multi-fichiers 2/4



Compilation des fichiers `hello.c` et `main.c` précédents

UNIVERSITÉ CÔTE D'AZUR

Exemple 2: Programme multi-fichiers 3/4

- ✓ **En fait, le compilateur est assez laxiste:**

```
main() {                          /* Fichier main.c */
    say_hello();
}
```

```
void say_hello(void) {            /* Fichier hello.c */
    printf("Hello, world!\n");
}
```

- ✓ **Dans** `main.c`**:**
  - **pas de déclaration de la fonction externe**
- ✓ **Dans** `hello.c`**:**
  - **pas d'inclusion du fichier** `<stdio.h>`
  - **déclaration de la fonction "simplifiée"**
- ✓ **La compilation arrive à son terme sans erreur**

UNIVERSITÉ CÔTE D'AZUR

Exemple 2: Programme multi-fichiers 4/4

- ✓ Le compilateur *gcc* peut nous aider à trouver les problèmes:
  - utiliser l'option de compilation `-Wall` pour voir tous les *warnings*
  - utiliser aussi l'option `-std=c99` pour être sûr de compiler du C99

```
$ gcc -Wall -std=c99 -o hello hello.c main.c
hello.c:1:1: warning: return type defaults to 'int' [enabled by default]
hello.c: In function 'say_hello':
hello.c:3:1: warning: implicit declaration of function 'printf' [-Wimplicit-function
hello.c:3:1: warning: incompatible implicit declaration of built-in function 'printf
hello.c:4:1: warning: control reaches end of non-void function [-Wreturn-type]
main.c:1:1: warning: return type defaults to 'int' [enabled by default]
main.c: In function 'main':
main.c:2:1: warning: implicit declaration of function 'say_hello' [-Wimplicit-functi
$
```

UNIVERSITÉ CÔTE D'AZUR

Exemple 3: Retour sur Hello World

- ✓ *Si on compile le hello world avec l'option* `-Wall`
  - *Il faut un type de retour* `int` *à la fonction* `main`
  - *la fonction* `main` *doit renvoyer une valeur (un entier)*

```
#include <stdio.h>
int main()
{
      printf("Hello, world!\n");
      return 0;
}
```

- ✓ Cette version est correcte

UNIVERSITÉ CÔTE D'AZUR

Exemple 4: La commande `cat` 1/2

```
#include <stdio.h>
int main()
{
     int c;
     c = getchar();
     while (c != EOF) {
          putchar(c);
          c = getchar();
     }
     return 0;
}
```

UNIVERSITÉ CÔTE D'AZUR

Exemple 4: La commande `cat` 2/2

- ✓ Ce programme
  - n'utilise pas le *style* C
  - Ne profite pas du fait que C est un langages d'expressions

```c
#include <stdio.h>
int main()
{
  int c;

  c = getchar();
  while (c != EOF) {
    putchar(c);
    c = getchar();
  }
  return 0;
}
```

```c
#include <stdio.h>
int main()
{
  int c;

  while ((c = getchar()) != EOF) {
    putchar(c);
  }


  return 0;
}
```

UNIVERSITÉ CÔTE D'AZUR

Exemple 5: Comptage de caractères 1/2

```
#include <stdio.h>
int main()
{
  long nbchar = 0;

  while (getchar() != EOF)
    nbchar++;
    printf("I've read %ld characters\n", nbchar);

  return 0;
}
```

UNIVERSITÉ CÔTE D'AZUR

Exemple 5: Comptage de caractères 2/2

```
#include <stdio.h>
int main()
{
  long nbchar = 0;

  while (getchar() != EOF)
    nbchar++;
    printf("I've read %ld characters\n", nbchar);

  return 0;
}
```

```
#include <stdio.h>
int main()
{
  long nbchar = 0;

  for (nbChar = 0; getchar() != EOF ; nbchar++) {
    /* Rien à faire */
  }
  printf("I've read %ld characters\n", nbchar);

  return 0;
}
```

UNIVERSITÉ CÔTE D'AZUR

Exemple 6: Equation du second degré

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

int main() {
  double a, b, c, delta;

  scanf("%lf %lf %lf", &a, &b, &c);
  delta = b*b - 4*a*c;

  if (delta < 0)
    printf("No real solution\n");
  else
    if (delta == 0)
      printf("Unique solution %.5f\n", -b/(2*a) );
    else {
      double sqr_delta = sqrt(delta);
      printf("Two solutions %g and %g\n",
             (-b - sqr_delta) / (2*a),
             (-b + sqr_delta) / (2*a));
    }
  return 0;
}
```