Programmation Procédurale – Langage C: introduction

Polytech'Nice Sophia Antipolis

Erick Gallesio

2015 – 2016

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 $\rightarrow$ 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)*

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

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)

Évaluation du langage (1/3)

Avantages

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

$\Rightarrow$ des millions de lignes écrite en C

Évaluation 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.
  $\Rightarrow$ 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, ...

Évaluation du langage (3/3)

Inconvénients

- Syntaxe à deux niveaux (pré-processeur)
- Grande liberté du programmeur ($\Rightarrow$ 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

Implémentations du langage C

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

Langage et normes (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
  - ...

Langage et normes (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

Et maintenant, quelques exemples. . .

Exemple 1: Hello world

Le classique **hello world**

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

Compilation:

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

Exécution:

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

Exemple 2: Programme multi-fichiers (1/4)

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

main() {
   say_hello();
}
```

```c
#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
```

Exemple 2: Programme multi-fichiers (2/4)

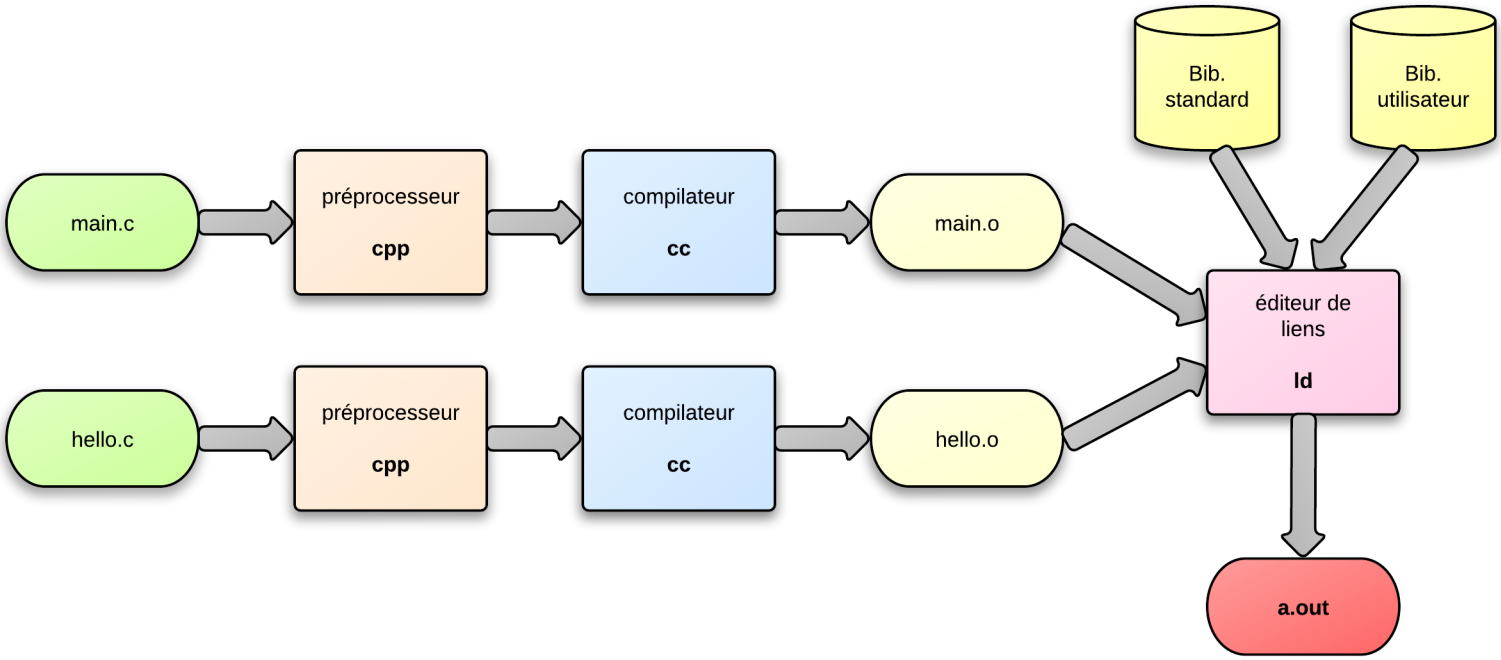

Figure : Processus de compilation

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

- Préprocesseur: **cpp**
- Compilateur: **cc**
- Éditeur de liens: **ld**

Exemple 2: Programme multi-fichiers (3/4)

En fait, le compilateur est assez laxiste:

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

```c
say_hello() {                           /* 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**

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 d
hello.c: In function 'say_hello':
hello.c:3:1: warning: implicit declaration of function 'printf' [
hello.c:3:1: warning: incompatible implicit declaration of built-
hello.c:4:1: warning: control reaches end of non-void function [-
main.c:1:1: warning: return type defaults to 'int' [enabled by de
main.c: In function 'main':
main.c:2:1: warning: implicit declaration of function 'say_hello'
$
```

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)

$\Rightarrow$

```c
#include <stdio.h>

int main()
{
   printf("Hello, world!\n");
   return 0;
}
```

Cette version est correcte.

Exemple 4: La commande cat (1/2)

```c
#include <stdio.h>

int main()
{
  int c;

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

  return 0;
}
```

Exemple 4: La commande cat (2/2)

Ce programme

- n'utilise pas le *style C*
- ne profite pas du fait que C est un langage d'expressions

```c
#include <stdio.h>

int main()
{
  int c;

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

```c
int main()
{
  int c;

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

  return 0;
}
```

Exemple 5: comptage de caractères (1/2)

```c
#include <stdio.h>

int main()
{
  long NbChar = 0;

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

  return 0;
}
```

Exemple 5: comptage de caractères (2/2)

```c
#include <stdio.h>

int main() {
  long NbChar = 0;

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

  return 0;
}
```

```c
#include <stdio.h>

int main() {
  long NbChar;

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

  return 0;
```

Exemple 6: équation du second degré

```c
#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;
}
```