Programmation C

J.-F. Lalande

15 novembre 2012

Résumé

Ce cours tente de couvrir tous les aspects liés au développement en langage C. Le cours débute par l'étude du langage C en lui même et se poursuit par l'étude des aspects spécifiques des tableaux et pointeurs. Ensuite, l'introduction des structures combinée à la notion de pointeurs permet d'étudier l'implémentation de structures de données complexes. Le cours termine par quelques éléments périphériques (compilateur, ncurses, ...).

Livre de référence

Disponible à la bibliothèque :

– Langage C, Claude DELANNOY, Eyrolles, 2005.

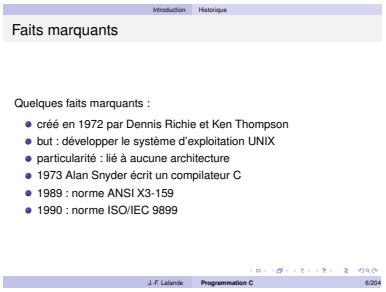
Table des matières

1 Introduction

1.1 Historique

Faits marquants

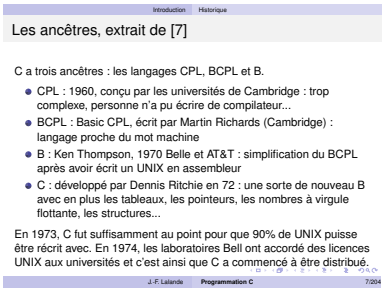
Quelques faits marquants :
- créé en 1972 par Dennis Richie et Ken Thompson
- but : développer le système d'exploitation UNIX
- particularité : lié à aucune architecture
- 1973 Alan Snyder écrit un compilateur C
- 1989 : norme ANSI X3-159
- 1990 : norme ISO/IEC 9899



Les ancêtres, extrait de [?]

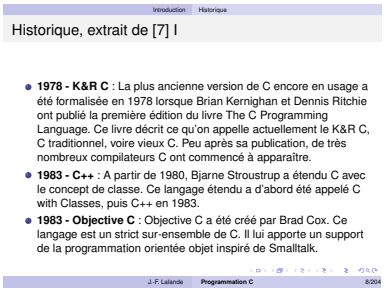
C a trois ancêtres : les langages CPL, BCPL et B.
- CPL : 1960, conçu par les universités de Cambridge : trop complexe, personne n'a pu écrire de compilateur...
- BCPL : Basic CPL, écrit par Martin Richards (Cambridge) : langage proche du mot machine
- B : Ken Thompson, 1970 Belle et AT&T : simplification du BCPL après avoir écrit un UNIX en assembleur
- C : développé par Dennis Ritchie en 72 : une sorte de nouveau B avec en plus les tableaux, les pointeurs, les nombres à virgule flottante, les structures...

En 1973, C fut suffisamment au point pour que 90% de UNIX puisse être récrit avec. En 1974, les laboratoires Bell ont accordé des licences UNIX aux universités et c'est ainsi que C a commencé à être distribué.



Historique, extrait de [?]

- **1978 - K&R C** : La plus ancienne version de C encore en usage a été formalisée en 1978 lorsque Brian Kernighan et Dennis Ritchie ont publié la première édition du livre The C Programming Language. Ce livre décrit ce qu'on appelle actuellement le K&R C, C traditionnel, voire vieux C. Peu après sa publication, de très nombreux compilateurs C ont commencé à apparaître.
- **1983 - C++** : A partir de 1980, Bjarne Stroustrup a étendu C avec le concept de classe. Ce langage étendu a d'abord été appelé C with Classes, puis C++ en 1983.
- **1983 - Objective C** : Objective C a été créé par Brad Cox. Ce langage est un strict sur-ensemble de C. Il lui apporte un support de la programmation orientée objet inspiré de Smalltalk.
- **1989 - ANSI C** : Un comité de standardisation a été créé en 1983 pour éviter que les quelques ambiguïtés et insuffisances du K&R C ne conduisent à des divergences importantes entre les compilateurs. Il a publié en 1989 le standard appelé ANSI C.
- **1998 - Standard C++** : C++ a évolué très longtemps. Ce n'est qu'en 1998, 8 ans après la création d'un comité, que le standard ISO C++ a été officiellement publié. Ce standard est tellement complexe (et légèrement incohérent), qu'en 2003, le compilateur GCC ne le met pas complètement en oeuvre, et ce n'est pas le seul.
- **1999 - C99** : Le dernier né de l'histoire est C99 (standard ISO de 1999) qui est une petite évolution de l'ANSI C de 1989. Les évolutions ne sont pas compatibles avec C++ et n'ont pas attiré beaucoup d'intérêt.



1.2 Premier principes

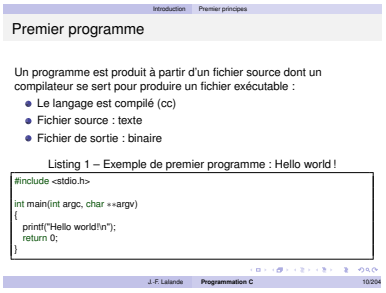
Premier programme

Un programme est produit à partir d'un fichier source dont un compilateur se sert pour produire un fichier exécutable :
- Le langage est compilé (cc)
- Fichier source : texte
- Fichier de sortie : binaire

Listing 1 – Exemple de premier programme : Hello world !

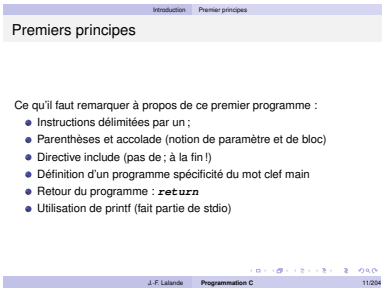
```
#include <stdio.h>

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



Premiers principes

Ce qu'il faut remarquer à propos de ce premier programme :
- Instructions délimitées par un ;
- Parenthèses et accolade (notion de paramètre et de bloc)
- Directive include (pas de ; à la fin !)
- Définition d'un programme spécificité du mot clef main
- Retour du programme : ***return***
- Utilisation de printf (fait partie de stdio)

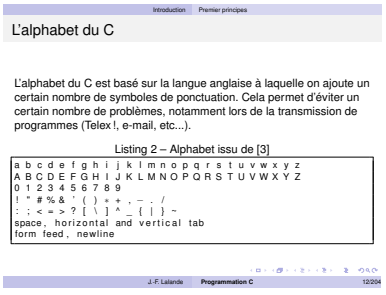


L'alphabet du C

L'alphabet du C est basé sur la langue anglaise à laquelle on ajoute un certain nombre de symboles de ponctuation. Cela permet d'éviter un certain nombre de problèmes, notamment lors de la transmission de programmes (Telex !, e-mail, etc...).

Listing 2 – Alphabet issu de [**?**]

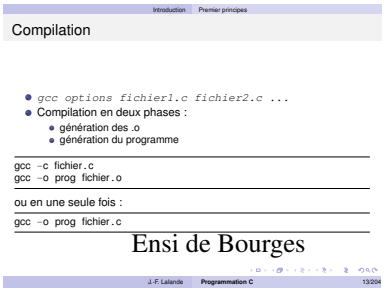
```
a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9
! " # % & ' ( ) * + , - . /
: ; < = > ? [ \ ] ^ _ { | } ~
space, horizontal and vertical tab
form feed, newline
```



Compilation

- *gcc options fichier1.c fichier2.c ...*
- Compilation en deux phases :
  - génération des .o
  - génération du programme

```
gcc -c fichier.c
gcc -o prog fichier.o
```



ou en une seule fois :

```
gcc -o prog fichier.c
```

Compilation (2)

Attention, les espaces importent !

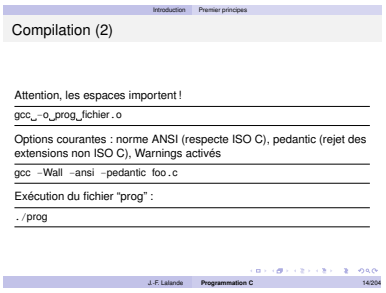
```
gcc␣-o␣prog␣fichier.o
```

Options courantes : norme ANSI (respecte ISO C), pedantic (rejet des extensions non ISO C), Warnings activés

```
gcc -Wall -ansi -pedantic foo.c
```

Exécution du fichier “prog” :

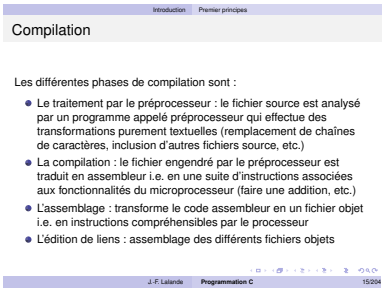
```
./prog
```



Compilation

Les différentes phases de compilation sont :

- Le traitement par le préprocesseur : le fichier source est analysé par un programme appelé préprocesseur qui effectue des transformations purement textuelles (remplacement de chaînes de caractères, inclusion d’autres fichiers source, etc.)
- La compilation : le fichier engendré par le préprocesseur est traduit en assembleur i.e. en une suite d’instructions associées aux fonctionnalités du microprocesseur (faire une addition, etc.)
- L’assemblage : transforme le code assembleur en un fichier objet i.e. en instructions compréhensibles par le processeur
- L’édition de liens : assemblage des différents fichiers objets



1.3 Types et variables

Variables : qu’est ce que c’est ?

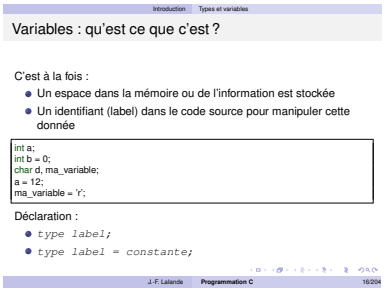
C’est à la fois :

- Un espace dans la mémoire ou de l’information est stockée
- Un identifiant (label) dans le code source pour manipuler cette donnée

```
1 int a;
2 int b = 0;
3 char d, ma_variable;
4 a = 12;
5 ma_variable = 'r';
```

Déclaration :

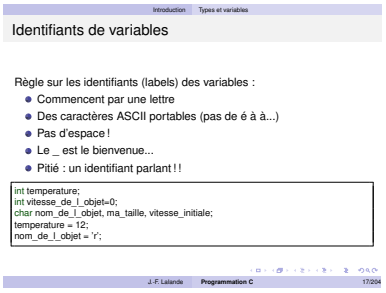
- *type label;*
- *type label = constante;*



Identifiants de variables

Règle sur les identifiants (labels) des variables :
- Commencent par une lettre
- Des caractères ASCII portables (pas de é à à...)
- Pas d'espace !
- Le _ est le bienvenue...
- Pitié : un identifiant parlant ! !

```
int temperature;
int vitesse_de_l_objet=0;
char nom_de_l_objet, ma_taille, vitesse_initiale;
temperature = 12;
nom_de_l_objet = 'r';
```



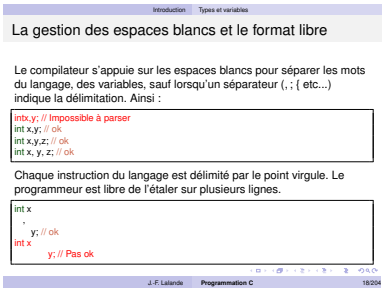
La gestion des espaces blancs et le format libre

Le compilateur s'appuie sur les espaces blancs pour séparer les mots du langage, des variables, sauf lorsqu'un séparateur (, ; { etc...) indique la délimitation. Ainsi :

```
intx,y; // Impossible à parser
int x,y; // ok
int x,y,z; // ok
int x, y, z; // ok
```

Chaque instruction du langage est délimité par le point virgule. Le programmeur est libre de l'étaler sur plusieurs lignes.

```
int x
  ,
    y; // ok
int x
        y; // Pas ok
```



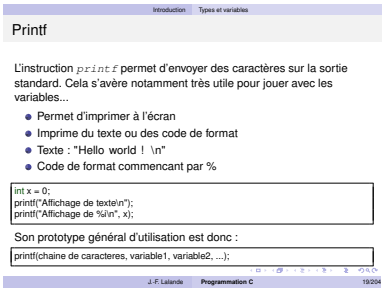
Printf

L'instruction `printf` permet d'envoyer des caractères sur la sortie standard. Cela s'avère notamment très utile pour jouer avec les variables...
- Permet d'imprimer à l'écran
- Imprime du texte ou des code de format
- Texte : "Hello world ! \n"
- Code de format commencant par %

```
int x = 0;
printf("Affichage de texte\n");
printf("Affichage de %i\n", x);
```

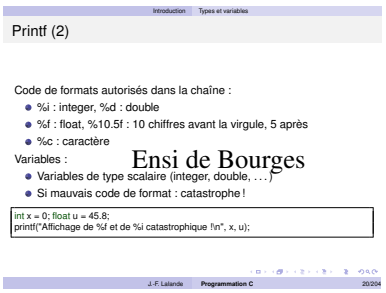
Son prototype général d'utilisation est donc :

```
printf(chaine de caracteres, variable1, variable2, ...);
```



Printf (2)

Code de formats autorisés dans la chaîne :
- %i : integer, %d : double
- %f : float, %10.5f : 10 chiffres avant la virgule, 5 après
- %c : caractère



Ceci est un exemple, cliquez sur le lien de téléchargement pour obtenir le cours complet.