Introduction à la programmation en C

Cours 1
19/12/2012

La compilation : du langage haut niveau au langage machine.

Samy BLUSSEAU, Miguel COLOM

Objectifs du cours :

- Être capable de construire des programmes en C
- De qualité
- De façon autonome
- Dans un environnement GNU/Linux
- En comprenant la machine

Déroulement :

Nous alternerons **pratique et théorie**, en développant des **projets** de traitement du signal (son, image...).

Pourquoi en C ?

- **Les programmes s'exécutent** aussi **rapidement** que possible
- C'est un langage **standard** (ISO/IEC 9899:1990)
- On peut construire un **programme en C pour presque n'importe quelle machine** programmable (téléphone portable, tablettes, robot Curiosity...)
- Les **programmes** sont **pérennes** (on peut toujours exécuter un programme en C d'il y a 15 ans, mais pas un programme Matlab vieux de 5 ans)

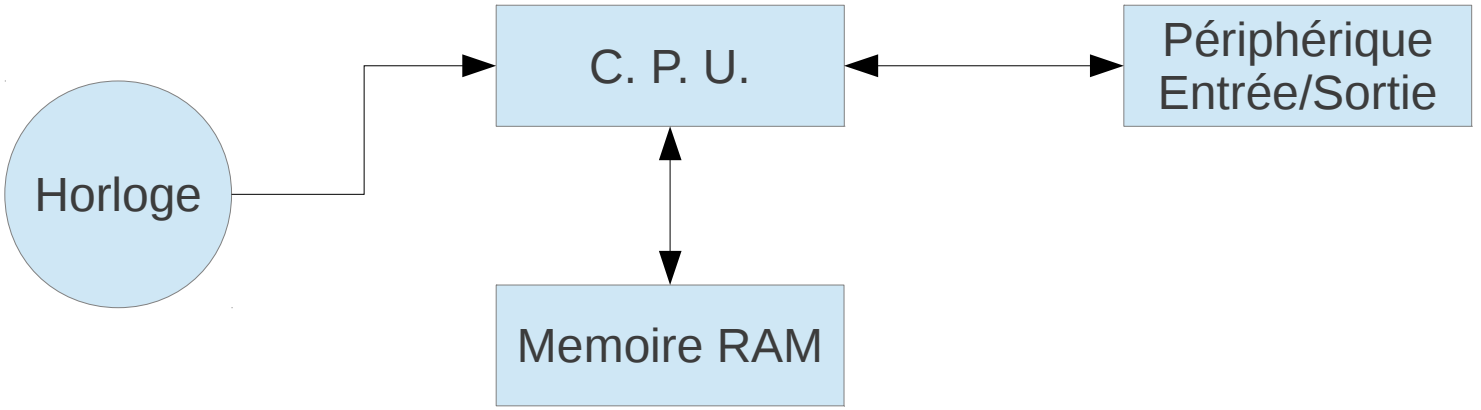
Comprendre la machine

- **CPU :** « Central Processing Unit », ou processeur (Pentium, ARM, Intel Core 2 Duo...)
  Se charge de lire des instructions et des données inscrites dans la mémoire RAM sous forme de séquences de 0 et de 1

- **Mémoire RAM :** « Random Access Memory »
  Stocke données et instructions servant aux programmes

- **Périphérique :** Ce qui permet la communication entre la machine et l'extérieur (souris, clavier, écran, E/S USB...)

- **Horloge :** A chaque cycle d'horloge, le CPU réalise une opération (le CPU exécute les instructions en un ou plusieurs cycles d'horloge, selon leurs complexité)



Langages haut/bas niveau

- **Le CPU en comprend que des séquences binaires** (« 011001010... »)
  Il s'agit du « langage machine » : il est non ambigu.
  C'est le langage plus « bas niveau » qui soit, car le plus proche de la machine

- Le langage naturel est le **langage** de plus **haut niveau**, car le plus **proche de nous** et le plus éloigné de la machine.
  Il est ambigu ! (Ex : « Alice et Bob ont dit à Carl qu'ils ont gagné au Loto ».
  → À qui se réfère « ils » ?)

- Pour **communiquer avec la machine**, il nous faut donc

  1) Un langage non ambigu (pour être compris de la machine), mais d'assez haut niveau (pour être compréhensible par nous) → un **langage de programmation**

  2) Un traducteur de ce langage vers le langage machine → le **compilateur**

- Exemple : Un compilateur de langage C, pour une machine avec processeur Pentium 4

Exemples

Langage humain :

« Faire la somme de 5 et 3 »

Langage haut niveau :

```
int a = 5;
int b = 3 ;

int c = a+b ;
```

Langage machine :

0100100011101010

Le compilateur

C'est un programme. Il procède en **2 phases indépendantes**:

1) Analyse du code source

- **Analyse lexicale :** Vérifie que tous les « mots » font partie du langage
  (Ex d'erreurs: « wile » au lieu de « while », « 3..14 » au lieu de « 3.14 ») → orthographe, vocabulaire

- **Analyse syntaxique :** Vérifie que les structures sont écrites selon la grammaire du langage
  (Ex. d'erreur : « if (x == 0) y = 4 **else else** y = 7; »)

- **Analyse sémantique :** vérifie que le code ait un sens (Ex. d'erreur : utiliser la valeur d'une variable qui n'a pas encore été assignée.)

  Au terme de cette première phase, un **code intermédiaire** est généré : un code **objet**, entre code haut niveau et code machine.
  Les instructions codées ne dépendent plus du langage haut niveau utilisé, et sont encore assez génériques pour être indépendantes de la machine qui les exécutera.

2) Génération de code exécutable (code machine)

Le code objet est traduit vers le code machine du CPU que l'on va utiliser pour exécuter le programme.

Avant la pratique : un rappel sur les **commandes GNU/Linux** :

- **man :** « Manual », pour voir la documentation relative à une commande (syntaxe, options...)
  ex. : man mkdir , puis « q » pour quitter.

- **pwd :** « Print Working Directory », pour voir où l'on se trouve dans l'arborescence des répertoires

- **cd** : « Change Directory », pour changer le répertoire courant

- **ls : «** List », pour voir la liste des fichiers et répertoires contenus dans un répertoire
  ls -l : plus de détails sur (taille, date de dernière modification des fichiers...)

- **mkdir :** « make directory » crée un répertoire dans le répertoire courant

- **mv :** « move », pour déplacer ou renommer un répertoire ou un fichier

- **cp :** « copy » pour copier des fichiers et répertoires

Quelques syntaxes :

- **cp /un/repertoire/un_fichier .**
  copier le fichier « un_fichier » du répertoire « /un/repertoire/ », dans le répertoire courant, désigné par le point « . »

- **mv proGGram.c program.c**
  change le nom « proGGram.c » en « program.c »

- **mv /repertoire1/mon_fichier /repertoire2/**
  Déplace « mon_fichier » de /repertoire1 vers /repertoire2

- **cd /repertoire/sous_repertoire**
  le répertoire courant devient /repertoire/sous_repertoire

- **cd ..**
  pour se placer dans le répertoire père, représenté par les deux points « .. »

- **/repertoire/du/programme/prog**
  exécute le programme « prog », qui se trouve dans le répertoire /repertoire/du/programme/
  Pour exécuter un programme, il faut toujours préciser son répertoire.
  Si « prog » se trouve dans le répertoire courant, il suffit de faire

./prog

Exercice 1 : compilons !

Télécharger le fichier « prog.c ».
(http://mcolom.perso.math.cnrs.fr/cours_C/src/prog.c),
L'ouvrir, et essayer de **générer le code objet** associé, grâce à la commande suivante
(où **« gcc »** est le **G**NU **C C**ompiler**)** :

gcc -c prog.c

Expliquer les erreurs, les corriger, puis essayer de générer le code objet à nouveau (même commande).

Quand plus aucun message d'erreur ne s'affiche, vérifier qu'un fichier « prog.o » a été créé, en tapant la commande :

ls

Exercice 1 (suite)

Générer le fichier exécutable « prog », grâce à la commande :

gcc prog.o -o prog

Cette commande traduit le code objet « prog.o » en code exécutable « prog » (l'option « -o » signifie « output »).
Exécuter le programme « prog » en tapant

./prog

Remarque :
Ici les deux étapes de la compilation peuvent se réaliser en une seule commande :

gcc prog.c -o prog

Exercice 2 : Votre premier programme en C !

Faire un programme qui prenne en argument une année (un nombre entier), et détermine si l'année est bissextile(*) ou non .

Pour cela, télécharger puis ouvrir le fichier leap.c (http://mcolom.perso.math.cnrs.fr/cours_C/src/leap.c).
Le code source est incomplet :
La fonction « **is_leap_year** » est vide.
La compléter de manière à ce qu'elle renvoie 1 si l'année d'entrée est bissextile, 0 sinon.
Vous pouvez vous aider du fichier « exemples_syntaxes.c » (http://mcolom.perso.math.cnrs.fr/cours_C/src/exemples_syntaxes.c)

A mesure que vous complétez le programme, n'oubliez pas d'**enregistrer**, **compiler**, **exécuter** !

(*) Rappel : Une année est bissextile si elle est divisible par 4 **ET** si, soit elle est divisible par 400, soit elle n'est pas divisible par 100.