TP0 – Découverte de l'environnement

Projet de programmation M1

15 Septembre 2015

Tous les TD/TPs que nous ferons ensemble se trouveront sur ma page :

`http://webusers.imj-prg.fr/~florent.capelli/`

avec les corrections (et les couleurs !). Vous y trouverez aussi ceux de l'an dernier, si vous en voulez encore. Si vous avez une quelconque question, vous pouvez me contacter à l'adresse : **fcapelli@math.univ-paris-diderot.fr**. Concernant le cours, vous trouverez tout sur la page de Roberto Amadio :

`http://www.pps.univ-paris-diderot.fr/~amadio/Ens/Algo/`

1 Introduction

1.1 Le langage C

Le langage que nous utiliserons pendant tout le cours est le langage de programmation C créé en 1972. Il en existe beaucoup d'autres (Pascal, Python, BASIC, Ocaml, C++, etc.) qui ont tous leurs propres spécificités, avantages et inconvénients (la syntaxe diffère, les opérations de bases aussi) mais qui au final ont tous la même fonctionnalité : vous permettre d'écrire des programmes que vous pourrez exécuter.

Le langage C est un langage simple et complexe à la fois. Simple parce qu'il a un fonctionnement assez proche de celui de la machine et ne propose donc que des éléments basiques de programmation. Complexe car c'est alors au programmeur de gérer certains aspects qui sont automatiquement pris en charge dans des langages plus récents. Cette proximité avec la machine en fait un des langages qui produit les programmes les plus rapides à l'exécution, à condition que le programmeur l'ait bien construit. Cette dépendance en l'abilité du programmeur en font aussi un langage plus exposé aux bugs et aux problèmes de sécurité que d'autres, plus récents.

Dans le cours et le TP, nous présenterons différents aspects de la programmation et de l'algorithmique que nous appliquerons ensuite en C. Cependant vous retrouverez plus tard dans pratiquement tous les autres langages que vous apprendrez une structure similaire.

1.2 Compilateur

Le C est un langage compilé, c'est-à-dire qu'il existe des programmes appelés *compilateurs* qui transforment un fichier texte, le *fichier source* ou le *source*, contenant du code C en un programme que peut exécuter la machine. Le programme résultant de cette opération dépend bien entendu de la machine sur lequel on souhaite l'exécuter (votre ordinateur personnel, le processeur d'une voiture, votre téléphone etc.). Un même code C pourra être exécuté sur plusieurs machines à condition d'avoir été compilé pour cette machine.

Il existe plusieurs compilateurs différents pour C qui ont chacun leurs spécificités et qui pour un même code et une même machine, ne produiront pas exactement le même programme, même si tous les programmes produits auront le même comportement – celui spécifié par le source, écrit en C. On peut citer, par exemple :

— GCC
— Borland Turbo C
— Visual C++ Express
— CompCert

Dans ce cours, nous utiliserons *uniquement* GCC, qui peut être installé gratuitement sur toute plateforme (GCC est sous licence GPL).

1.3 L'éditeur de texte

Pour taper votre programme, il faut utiliser un éditeur de texte. Nous ne parlons pas ici de Word ou OpenOffice qui produisent des documents contenant bien plus d'informations que le simple texte brut dont le compilateur a besoin (un document Word va contenir la mise en page du document par exemple). Sous Windows par exemple, le bloc-note serait théoriquement suffisant pour écrire nos programmes mais il est un petit peu trop rustique : on préfére choisir un éditeur avec des fonctionnalités supplémentaires qui facilitent le travail du programmeur. Une fonctionnalité importante est la *coloration syntaxique* : l'éditeur est capable d'afficher votre code avec des couleurs qui le rendront plus lisible : les mot-clés en gras, les chaînes de caractères dans une autre couleurs, etc.

Si le choix du compilateur influence le programme obtenu à la fin, l'éditeur de texte lui n'y change rien. Vous êtes donc libres de choisir celui que vous préférez et avec lequel vous êtes le plus à l'aise. Si vous n'en connaissez aucun, sur les machines Linux de l'université, nous vous conseillons d'utiliser *gedit* (vous pouvez aussi l'installer sous Windows et MacOS chez vous `https://wiki.gnome.org/Apps/Gedit`). On peut aussi citer sous Linux :

— gedit (conseillé pour les débutants qui ne savent pas vraiment quoi choisir)
— emacs (si vous voulez un éditeur qui fait tout sauf le café)
— vim (interdit aux moldus)

Sous Mac :

— Xcode
— Aquamacs

Sous Windows :

— Notepad++

2 Bonjour monde !

Pendant cette séance, on va se concentrer sur un des programmes les plus simples, le programme Hello World qui affiche un texte à l'écran. Le voici :

```c
#include <stdio.h>

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

Détaillons rapidement ce programme. On identifie clairement la ligne contenant `printf` qui va se charger d'afficher du texte à l'écran. À quoi sert le reste ? C'est le minimum vital pour qu'un programme C s'exécute. Le bloc `main`, entre deux accolades, est présent dans tout programme C. C'est la *fonction principale*, la première chose qu'exécutera votre programme. Le `return 0` permet simplement d'envoyer 0 à votre système d'exploitation lorsque le programme se termine. Cela indique que tout s'est bien passé. Enfin, le `#include <stdio.h>` indique qu'on veut avoir accès à des commandes définie dans le fichier `stdio.h`. C'est une *librairie* classique en C, qu'on utilisera de façon presque systématique puisqu'elle contient les fonctions pour afficher du texte. Il en existe d'autres : on pourrait vouloir utiliser la fonction `cosinus` par exemple. Elle ne se trouve pas par défaut dans le langage C, mais dans une librairie appelée `math.h`. Pour l'utiliser, il faudrait ajouter `#include <math.h>` en début de fichier. Nous introduirons plus tard les librairies classiques du C en fonction de nos besoins.

On remarque aussi que chaque instruction est terminée par un point-virgule. C'est très important. On pourrait ne pas sauter de ligne du tout et le programme serait encore valable (mais illisible). Cependant, enlever le point-virgule rendrait le programme invalide et on ne pourrait pas le compiler.

Exercice 1. Ouvrez votre éditeur de texte préféré.

1. Assurez-vous de savoir : ouvrir/enregistrer un fichier, rechercher un mot dans le texte, remplacer toutes les occurences d'un mot par un autre avec votre éditeur.
2. Créez un nouveau fichier `hello.c` dans lequel vous taperez le programme Hello World.

En général, on nomme un fichier C avec l'extension `.c`. Cela aide les éditeurs à bien colorer votre syntaxe et indique aux autres gens ce que contient votre fichier, mais ce n'est pas une obligation en soit.

On respectera cette convention. Aussi, les fichiers librairies sont généralement en `.h`, h comme header (en-tête).

Respectez les espaces : vérifiez bien que le printf et le return sont alignés, un peu en retrait par rapport aux accolades (une tabulation). C'est ce qu'on appelle l'*indentation* : on aligne les blocs qui vont ensemble. Cela améliore grandement la lisibilité du code. Il faut prendre dès maintenant les bonnes habitudes.

Exercice 2. [Phase terminal] Dans cet exercice, nous rappelons quelques commandes utiles dans un terminal et compilons notre premier programme ! Quand vous avez un doute sur une commande, tapez `man nom_de_la_commande` pour avoir des explications (cela marche aussi pour les fonctions standard du C).

1. Ouvrez un terminal. Observez qu'à gauche du curseur sont indiqués : le nom de l'utilisateur, le nom de l'ordinateur et le dossier courant. Par exemple : `bob@lamaison:~/documents$`. Le `~` est une abbréviation pour `/home/bob/`. Un `.` désigne le répertoire courant et `..` le répertoire parent. En utilisant la commande `cd` (pour *change directory*), naviguez dans l'arborescence des dossiers jusqu'au dossier où vous avez enregistré `hello.c`. Appuyez deux fois sur la touche tab pour autocompléter les noms de dossiers. Vous pouvez afficher les fichiers du dossier courant avec la commande `ls`.
2. Créez le dossier `tp0` avec la commande `mkdir tp0` et déplacez le fichier `hello.c` dans ce répertoire avec la commande `mv hello.c tp0/`. Allez dans le dossier `tp0`.
3. Compilez le fichier `hello.c` avec la commande `gcc -Wall hello.c`. Le `-Wall` indique qu'on veut afficher le plus d'informations possibles concernant les erreurs de compilation. Mieux vaut l'indiquer. Si tout se passe bien, un nouveau fichier `a.out` a été créé. C'est le programme compilé, un fichier *exécutable*. Vous pouvez l'exécuter en tapant `./a.out`. Si vous voulez changer le nom de l'exécutable que GCC produit, vous pouvez lui indiquer avec l'option `-o` au moment de la compilation : `gcc -Wall hello.c -o hello.out`. On exécutera alors le programme avec `./hello.out`.

   Astuce 1. *Vous pouvez faire défiler les dernières commandes utilisées en appuyant sur la flèche du haut.*

Exercice 3. Dans cet exercice, on va modifier `hello.c` pour changer son comportement.

1. Enlevez un point-virgule dans le code et réessayez de compiler. Essayer de comprendre ce que GCC vous affiche. Ce n'est pas la dernière fois que vous verrez ça. De même, essayez de supprimer la ligne `#include <stdio.h>` et observez la réponse de GCC.
2. Supprimez tous les saut-de-lignes du programme initial sauf le premier. C'est illisible mais essayez de compiler.
3. Changez le texte pour afficher un autre message. Vous pouvez sauter des lignes en utilisant `\n`. Par exemple : `"ceci est \n un message"` s'affichera sur deux lignes. Ajoutez d'autres `printf` avec d'autres messages. Observez le comportement. N'oubliez pas de re-compiler entre chaque changement.
4. À mesure que votre programme se complique, il sera intéressant d'anoter certains endroits du code pour vous rappeler de ce que vous aviez en tête à ce moment-là. On écrit des commentaires de deux façons :
   — Sur une seule ligne, en la faisant débuter par //
   — Sur plusieurs ligne, en encadrant le texte par /* et */.
   Les commentaires seront ignorés du compilateur mais vous aideront beaucoup lorsqu'il faudra reprendre des morceaux du programme que vous avez écrits il y a longtemps (expliquer le fonctionnement d'une fonction, le contenu d'une variable, ou simplement laisser une liste de ce qu'il reste à faire). Essayez d'ajouter des commentaires à votre code. Vous pouvez par exemple indiquer au début du fichier que ce code fait partie du TP 0, mettre votre nom et la date etc.

Exercice 4. Les variables sont des cases mémoires dans lesquelles vous pouvez ranger des informations au cours du programme. Elles ont un type en fonction des données qu'elles contiennent : texte, entier, nombre à virgule etc. On va manipuler dans cet exercice les variables contenant des entiers :

```c
#include <stdio.h>

int main() {
```

```c
  /* Declaration et initialisation de la variable.
     Type : integer = entier
   */
  int x = 1;
  int y = 5;

  // Operation arithmetique de base
  x = (3*x+5)/3;
  y = y-x;
  // Affichage
  printf("La valeur de x+y est %d+%d=%d.\n", x, y, x+y);
  return 0;
}
```

1. Recopier, compiler et exécuter le programme précédent. Déclarez une variable z juste après x et y et lui donner la valeur x+y. Afficher la valeur de z avant et après avoir changé la valeur de x et y. Que remarquez-vous ?
2. Essayer d'afficher la valeur d'une variable nommée foo sans la déclarer. Observez l'erreur donnée par GCC.

Exercice 5. Dans cet exercice, on va introduire doucement des opérateurs en C qui permettent de modifier le cours de l'exécution du programme. En exécutant les programmes suivants, votre but est de comprendre ce que fait chaque opérateur et d'être capable de modifier le programme pour avoir un nouveau comportement. Remarquez que ces opérateurs utilisent souvent des blocs d'instructions délimités par des accolades { et }.

1. On s'intéresse à l'opérateur `for`.

```c
#include <stdio.h>

int main() {
  int i = 0;
  for(i=3;i<10;i=i+1) {
    printf("Hello World\n");
  }
  return 0;
}
```

   Exécuter le programme et comprendre ce que fait `for`.
2. Modifier ce programme pour qu'il affiche votre prénom 10 fois.
3. Modifier ce programme pour qu'il affiche les entiers de 2 à 2015. Puis les entiers pairs de 2 à 2015.
4. On rappelle que *if* veut dire *si* en anglais, *then* veut dire *alors* et *else* veut dire *sinon*. Que fait ce programme ?

```c
#include <stdio.h>

int main() {
  int i = 0;
  for(i=0;i<10;i=i+1) {
    if (i == 5) {
      printf("a");
    }
    else {
      printf("b");
    }
  }
  return 0;
}
```

5. Modifiez ce programme pour qu'il affiche les entiers de 1 à 100 suivis d'un point d'exclamation s'ils sont divisibles par 3.
6. (Un peu plus dur) Créer un programme qui a une variable entière n et affiche la valeur de $(n!)$.