# info803 : informatique

# TP 1 : introduction au C

Pierre Hyvernat
Laboratoire de mathématiques de l'université de Savoie
bâtiment Chablais, bureau 22
téléphone : 04 79 75 94 22
email : `Pierre.Hyvernat@univ-savoie.fr`
www : `http://www.lama.univ-savoie.fr/~hyvernat/`

*Un petit compte-rendu de TP vous sera demandé en fin de séance. Le compte-rendu sera un petit fichier texte "*`nom-prenom.txt`*" dont le squelette est disponible sur ma page web. Seules les questions intéressantes demandent une réponse rédigée... En plus du fichier* `nom-prenom.txt`*, vous devrez me rendre les fichiers "*`.c`*" correspondants. Il vous faudra donc créer une archive en utilisant la commande*

```
> tar cf nom-prenom.tar fichier1.c ... nom-prenom.txt
```

*Demandez moi pour plus de détails...*

### Exercice 0 : prise en main de l'environnement Unix

Le langage C est le langage de Unix, et par conséquent, Unix est le système d'exploitation de C (??). La première étape est donc de (re)démarrer votre machine en choisissant le système Linux.* Une fois que l'ordinateur a démarré, il faut taper son login (`phyve` dans mon cas) puis son mot de passe. Vous devriez ensuite vous retrouver dans un environnement graphique. (Probablement "Gnome" si vous n'avez rien fait de particulier.)

*Question 1 :* prenez quelques minutes pour explorer le système, aller voir dans les menus, créer un répertoire "`info803`" dans votre dossier personnel etc. Utilisez un navigateur Internet (Firefox, ...) pour aller récupérer le fichier exemple de compte-rendu sur ma page Internet. Sauvegardez le fichier sous un nom approprié dans le répertoire `info803`.

*Question 2 :* les deux programmes importants dont nous nous serviront sont

- un éditeur de texte (Emacs, XEmacs, Gedit, ...)
- un "terminal" pour entrer des lignes de commande (compilation, exécution de nos programmes etc.)

Chercher ces programmes dans les menus et lancez les. Dans l'éditeur de texte, ouvrez le fichier de compte-rendu et modifiez les nom et prénom.
Dans le terminal, déplacez vous dans le répertoire `info803` avec la commande

```
> cd info803/
```

*Question 3 :* survivre dans un terminal. Les commandes de base à connaître pour utiliser le terminal sont

- `> ls` ("list") pour avoir la liste des fichiers présents dans le répertoire courant,
- `> cd nom-rep` ("change directory") pour changer le répertoire courant
- `> man prog` ("manual") pour obtenir l'aide en ligne du programme "`prog`"

Regardez l'aide en ligne de la commande "`mkdir`". À quoi sert cette commande ?
Regardez l'aide en ligne de la commande "`gcc`". Qu'en pensez-vous ?
Comment faire pour avoir l'aide en ligne de la commande "`man`" ?

* Linux est une variante gratuite de Unix...

**Exercice 1 : premiers programmes en C**

*Question 1 :* créez un nouveau fichier "`info803/exo-1.c`" dans l'"éditeur de texte. Écrivez dans ce fichier le programme "`hello-world`" vu en cours.
Pour compiler le programme, utilisez les commandes suivantes dans le terminal :

- `> pwd` ("print working directory") pour vérifier que vous êtes bien dans le bon répertoire. (La commande doit écrire "`...info803`".)
- Si vous n'êtes pas dans le bon répertoire, utilisez la commande `> cd info803` pour se positionner dans le bon répertoire, et recommencez à l'étape précédente.
- Utilisez la commande `> ls` pour vérifier qu'il y a bien un fichier `exo-1.c` dans le répertoire.
- Compilez le programme avec la commande "`gcc -Wall exo-1.c -o exo-1`".
- Vérifier l'existence d'un fichier `exo-1` dans le répertoire.
- Exécutez le programme `exo-1` avec la commande `> ./exo-1`

*Question 2 :* modifiez le programme pour qu'il déclare deux variables de type entier et affiche le message suivant

`La valeur de 3 plus 4 est égale à 7.`

(Bien entendu, si les valeurs des deux variables sont autres chose que 3 et 4, le programme devra afficher les bonnes valeurs...)

*Question 3 :* si `n` est une variable de type entier, vous pouvez lire un entier du clavier dans un programme C en utilisant l'instruction

`scanf("%i",&n) ;`

Suite à cette instruction (et si l'utilisateur à réellement rentré un entier au clavier), la valeur de `n` sera l'entier tapé par l'utilisateur.

En utilisant l'instruction `scanf`, modifiez votre programme `exo-1.c` pour qu'il demande les deux entiers dont il doit calculer la somme. Expérimentez...

*Question 4 :* modifier le fichier `exo-1.c` pour qu'il demande un nombre $n$, et calcule ensuite la moyenne des $n$ entiers rentrés par l'utilisateur.

N'oubliez pas de joindre le fichier `exo-1.c` à votre archive de compte-rendu.

**Exercice 2 : calcul d'une puissance**

Nous allons maintenant essayer de calculer les puissances d'un nombre $x^n$ en utilisant les différentes méthodes vue en cours. Comme il faut que $n$ soit grand pour pouvoir constater les différences, nous allons en fait calculer les puissances *modulo en autre entier*. Ceci permettra d'éviter les débordements. (Quand un entier devient trop grand...)

Le but est donc de produire un programme qui demande les valeurs de $x$, $n$ et $p$ (tous des entiers) et calcule la valeur "$x^n \bmod p$" de différentes manières. Pour comparer l'efficacité, on affichera également le temps de calcul de chaque fonction.

Pour pouvoir afficher le temps de calcul, vous rajouterez les lignes suivantes dans votre fichier :

```
#include <sys/time.h>

/* cette fonction renvoie le nombre de microsecondes
 * depuis le 1 janvier 1970 (!!) */
long maintenant(void) {
  struct timeval start ;
  gettimeofday(&start, NULL) ;
  return (start.tv_sec * 1000000 + start.tv_usec) ;
}
```

Pour évaluer le temps pris par la fonction `puissance(x,n,p)`, on peut déclarer une variable "`t`" de type `long int` (grand entier) et utiliser les instructions suivantes

```
t : long int ;
...
t = maintenant() ;
resultat = puissance(x,n,p) ;
```

```
t = maintenant() - t ;
```
Suite à cette dernière instruction, `t` contient le nombre de micro-secondes nécessaires pour calculer `puissance(x,n,p)`.

*Question 1 :* dans le fichier `exo-2.c`, écrivez une fonction `puissance_rec` à trois arguments qui calcule "$x^n$ mod $p$" de manière récursive naïve. Écrivez ensuite la fonction `main` qui permet d'utiliser cette fonction sur des entiers choisis par l'utilisateur. Vous devez afficher le résultat ainsi que le temps nécessaire à l'exécution de votre fonction.

*Question 2 :* rajoutez une fonction `puissance_iter` qui calcul la même chose de manière itérative (avec une boucle `for`). Modifiez la fonction `main` pour qu'elle affiche le résultat et le temps utilisé pour les deux fonctions séparément.
Expérimentez...

*Question 3 :* rajoutez une fonction `puissance_chinoise_rec` qui utilise la méthode chinoise de manière récursive. Testez et concluez...

*Question 4 :* idem mais en utilisant une méthode chinoise itérative. Testez et concluez.

N'oubliez pas de joindre le fichier `exo-2.c` à votre archive de compte-rendu.

**Exercice 3 : Fibonacci, again...**

*Question 1 :* dans un fichier `exo-3.c`, écrivez la fonction `fib_rec` qui calcule la suite de Fibonacci de manière récursive naïve. Utilisez la fonction `main` pour demander un entier $n$ et affichez le résultat $F_n$. Testez et expérimentez. Qu'en pensez-vous ?

*Question 2 :* rajoutez une fonction `fib_iter` qui calcule la suite de Fibonacci de manière itérative efficace. Testez et expérimentez. Qu'en pensez-vous ?

*Question 3 :* rajoutez un compteur qui compte le nombre d'additions utilisées dans chacune de fonctions. Testez...

*Question 4 :* quelle est la plus grande valeur de $F_n$ que vous pouvez calculer avec la première fonction ? Avec la seconde ?
Comment faire pour pouvoir calculer des termes plus élevés avec la seconde fonction ? Jusqu'où pouvez vous calculer ? Qu'en pensez-vous ?

N'oubliez pas de joindre le fichier `exo-2.c` à votre archive de compte-rendu.

**Exercice 4 : multiplication de matrices**

*Question $\infty$ :* programmez au mieux l'algorithme vu en cours qui permet de minimiser le nombre de multiplications scalaires pour le calcul du produit d'une chaîne de matrices.

Spécifiez le problème et les limites de votre algorithme. Commencez par une version assez simple avant de le complexifier pour obtenir un algorithme le plus général possible.