MMFAI
Cours de programmation

TP numéro 1
semaine du 5 octobre 1998

Commencer :

Commencez par vous connecter (login, passwd, etc.) puis tapez la commande `mesg n` pour bosser tranquillement. Une bonne habitude est de lire son mail et le forum chaque jour en se connectant, mais pour gagner du temps en T.P., lancez directement un éditeur de texte (joe, vi, **emacs**, etc.).

Pour bien travailler, organisez votre écran avec une fenêtre pas trop large dédiée à l'éditeur de texte (en programmant, on n'utilise que rarement des lignes très longues), puis une fenêtre de terminal placée à coté ou en dessous de la première.

Un programme C est contenu dans un fichier dont l'extension est `.c`, par exemple `toto.c`
Le cycle de production pour programmer en C est le suivant:

- édition du programme C `toto.c` avec votre éditeur préféré
- compilation à l'aide du compilateur C (typiquement `gcc` ...)

  ```
  gcc -o toto toto.c
  ```

  (pour les curieux, l'option `-o toto` dit que l'executable s'appellera `toto` et non pas `a.out` qui est le nom par défaut... faire `man gcc` pour en savoir plus)
- exécution du programme: on tape `toto` et on regarde ce qui se passe

Au contraire donc de ce qui se passe en `CamlLight` ou `Maple` vous ne voyez pas de résultats avant d'avoir complété ce cycle.

Exercice 1 (Hello World)

Essayez le cycle de production précédent sur le programme suivant (attention, un ; de perdu et ça ne marche pas, mais vous pouvez vous passer des commentaires!):

```
/* ceci est un commentaire */
/* voyons comment imprimer */
#include <stdio.h>;

main()
{
printf("Bonjour\n");
        exit(0);
}
```

Remarquons que:

- les fichiers avec extension `.h` contiennent des définitions
- on peut inclure des définitions à l'aide de `#include`
- C dispose des plusieurs types de base et de mécanismes pour définir des types utilisateur. Pour le moment, contentons nous de savoir que `void`, `int`, `float` et `char` sont des types valides
- une déclaration de fonction est de la forme

```
TYPE_RESULTAT nom(par1, par2)
type1 par1;
type2 par2;
{
        ...
        return RESULTAT;
}
```

  ou bien, si on suit le standard ANSI,

```
TYPE_RESULTAT nom(type1 par1, type2 par2)
{
        ...
        return RESULTAT;
}
```

- *tout* en C est une fonction qui prend des paramètres et renvoie un résultat, *mais* on si on veut on peut ignorer ces résultats (cas de printf), et quand ces résultats ne sont jamais significatifs, le type du résultat est `void` (cas de exit). Faites donc `man exit` et `man printf`.

Exercice 2 (Imprimer avec un *format*)

Pour imprimer un valeur en C, il faut se servir de la fonction `printf`, qui attend une chaîne de caractères (le *format*) et un nombre arbitraire de valeurs (même 0 comme ci-dessus) qui seront placés dans le format aux positions indiquées par les *descripteurs de format* tels que `%d`, `%f` et `%s`. Modifiez votre programme comme suit et testez à nouveau.

```
#include <stdio.h>;

main()
{
printf("Bonjour, ceci (%d) est un entier, ceci (%f) est un flottant",300,3.1416);
printf("et ceci (%s) est une \"string\" (chaine de caracteres)\n","blah blah");
        exit(0);
}
```

Faite `man printf` pour plus d'information sur les *formats d'impression*.
Notez que `\n` indique un fin de ligne (de l'anglais "newline").
Comment imprime-t-on un caractère `%` ? Et un \ ?

Exercice 3 (Lire une valeur)

Pour lire une valeur sur l'entrée standard en C, il faut se servir de la fonction `scanf`, qui attend une chaîne de caractères (le *format*) et un nombre arbitraire de valeurs (même 0 comme ci-dessus) qui seront placés dans le format aux positions indiquées par les *descripteurs de format* tels que `%d`, `%f` et `%s`. Modifiez votre programme comme suit et testez à nouveau.

```
#include <stdio.h>;

main()
{
int i;
float f;
printf("Bonjour: rentrez un entier et un flottant separes par
                des espaces ou un retour chariot\n");
        scanf("%d %f ",&i,&f);
printf("Bonjour, ceci (%d) est un entier, ceci (%f) est un flottant",i,f);
        exit(0);
}
```

Rémarque: on a déclaré une *variable* `i` de type `int` et une `f` de type `float`. Comme en C le passage de paramètres aux fonctions se fait par valeur, on est obligé de donner à la fonction `scanf` l'*adresse* mémoire où sont physiquement alloués f et i. Cela se fait avec l'opérateur `&` qui renvoie un entier nommé *pointeur*, vu qu'il pointe sur une case mémoire.

Les instructions minimales

En C vous pouvez utilisez déjà les instructions suivantes:

- affectation:
  `i=3;`
- addition/soustraction de 1:
  `i++;i--;`
- comparaison:
  `< > <= >= == !=`
- arithmetique simple:
  `+ * - / %`
- séquence:
  `{ inst; inst; }`
- conditionnelle:
  `if (expr) inst;`
  `if (expr) inst; else inst;`
  `expr ? inst : inst`
- boucle while:
  `while (expr) inst;`
- boucle for:
  `for (start; end; cont) inst;`
  Exemple:
  `for (i = 0, j = 99; i < j; i++, j--) ...;`
- retour d'un résultat:
  `return(expr);`

Bien, vous avez maintenant le minimum indispensable pour pouvoir utiliser le langage C (on en apprendra plus au fur et à mesure de nos nécessités, mais pour les plus intéressés, il serait mieux de se procurer la référence du langage: *Kernigan, Ritchie. Le langage C.* Vous pouvez aussi trouver un résumé succinct dans `http://www.dmi.ens.fr/~beig/systeme/langageC.html`).

Passons donc aux choses sérieuses ...

Premiers programmes

Exercice 4 (Algorithme d'Euclide bête) *Écrivez un programme C* `eubete.c` *qui lit deux entiers* a, b *sur l'entrée standard et calcule et imprime* $pgcd(a, b)$ *en utilisant seulement la soustraction.*

Exercice 5 (Algorithme d'Euclide) *En sachant que en C les opérations* `+,*,/,-` *sur les entiers sont bien définies, écrivez maintenant un programme C* `eu.c` *qui lit deux entiers* a, b *sur l'entrée standard et calcule et imprime* $pgcd(a, b)$ *en utilisant l'algorithme vu en cours.*

Exercice 6 (Algorithme d'Euclide étendu) *Écrivez un programme C qui réalise l'algorithme d'Euclide étendu.*

Exercice 7 (Les nombres de Fibonacci) *Écrivez un programme qui lit un entier* n *et imprime* F_n*, le* n*-ième nombre de Fibonacci.*
Essayez avec 40*. Si au but de quelques secondes vous n'avez pas de résultat, arrêtez le programme (tapez Ctrl-C) et réfléchissez à votre algorithme.*

Exercice 8 (Exponentiel modulo) *Écrivez un programme qui lit trois entiers* a, b, n *et imprime* $(a^b) \bmod n$*. Pour cela, il peut être utile de savoir que C dispose d'opérateurs pour travailler sur les bits, comme* `<<` *et* `>>` *qui font du roulement a gauche ou à droite de la representation binaire d'un entier (par ex.* `1<<4` *= 16).*

N'oubliez pas de vous déconnecter avant d'abandonner le clavier (voir notes du stage Unix).