Prise en main du langage C

1. Création du fichier et compilation

Pour créer votre premier document C, ouvrez un éditeur de texte, par exemple Emacs (qui a l'avantage de présenter une coloration syntaxique).

Enregistrez le document sous le nom **helloworld.c** dans un répertoire de votre choix. Entrez le code suivant :

```
#include <stdio.h>
#include <stdlib.h>

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

Enregistrer à nouveau le document.

Ouvrez maintenant une fenêtre de terminal. Tout d'abord, vous devez vous placer dans le répertoire où se trouve votre document. Ceci se fait grâce à la commande `cd` (pour *change directory*). Par exemple, si votre document se trouve dans le répertoire `ALBA`, situé lui-même dans le répertoire `Documents`, vous pouvez y accéder en tapant dans le terminal la commande

```
cd Documents/ALBA/
```

puis en appuyant sur la touche Entrée.

Astuce : la touche de tabulation permet de compléter automatiquement le nom des répertoires après en avoir tapé les premiers caractères.

Pour pouvoir utiliser votre programme C, vous devez le compiler pour produire un exécutable. Pour cela, une fois que vous êtes dans le répertoire contenant votre document, tapez la commande

```
gcc -o helloworld helloworld.c
```

La commande `gcc` fait appel au compilateur C. L'option `-o helloworld` signifie que vous voulez créer un exécutable, dont le nom sera `helloworld`. Enfin, `helloworld.c` est le nom du fichier que vous voulez compiler. Ici, on a choisi de nommer l'exécutable de la même façon que le document C (à l'exception de l'extension), mais ce n'est pas obligatoire.

Enfin, pour exécuter le programme, il suffit de taper la commande

```
./helloworld
```

Qu'obtenez-vous ?

2. Explications de base sur le code

Décryptons le code C que nous avons tapé. Tout d'abord, les deux premières lignes

```
#include <stdio.h>
#include <stdlib.h>
```

ont pour rôle de charger des modules contenant les principales fonctions C. Il faut les taper au début de chaque document. La suite est le plus important :

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

Tout programme C doit contenir une fonction `int main()`. C'est cette fonction qui indique au compilateur ce qu'il doit faire. Elle ne prend aucun argument (d'où les parenthèses vides après le nom de la fonction `main`), et renvoie un entier (en l'occurrence, 0, d'où le `return 0 ;` à la fin). L'entier renvoyé a peu d'importance, celui-ci n'apparait pas à l'écran. En général, on renvoie 0 pour signaler que le programme s'est exécuté correctement.

Ce qui fait le coeur du programme, c'est ce qu'on l'on écrit entre le `int main() {` et le `return 0 ; }`. Dans notre code, il s'agit de

```
  printf("Hello world!\n");
```

La fonction `printf` permet d'imprimer quelque chose à l'écran. Elle prend en argument une chaîne de caractères entre guillemets ; `\n` est un caractère spécial qui désigne un « retour à la ligne ».

En général, un programme C contiendra d'autres fonctions en plus de la fonction `main`. Cela devient important pour structurer le programme et le rendre lisible dès qu'il devient complexe. On peut faire appel à ces fonctions les unes dans les autres, et en particulier dans la fonction `main`, mais c'est ce qui est dans la fonction `main` qui sera exécuté par le programme.

Exercice 1. Écrivez un programme qui affiche à l'écran le texte suivant (sauts de ligne compris) :

```
Bonjour.
Bienvenue au premier TP d'algorithmique de l'année scolaire
2011-2012.
```

3. Variables, tests, boucles

Comme dans tout langage de programmation, on peut faire des opérations algébriques, déclarer des variables, faire des boucles, des tests, etc. Ci-dessous, les syntaxes correspondantes.

Pour déclarer une variable, on indique le type de la variable puis son nom. Éventuellement, on l'initialise avec `=` suivi de la valeur. Exemple :

```
int n = 10;
```

On termine toujours une instruction par un point-virgule.

Pour déclarer une fonction, on écrit le type de ce qu'elle doit renvoyer, le nom de la fonction, puis entre parenthèses les types des arguments qu'elle prend suivis de leurs noms. Le contenu de la fonction est placé entre accolades. On indique ce qu'elle renvoie par le mot clé `return`. Par exemple, la fonction suivante renvoie la somme des entiers donnés en argument :

```
int somme (int n, int m) {
  return n+m;
}
```

Les principaux types sont : `int` pour les nombres entiers, `double` pour les nombres flottants (à virgule), `void` pour l'absence de type : c'est utile notamment si une fonction ne renvoie rien, et se contente d'effectuer une action telle qu'écrire quelque chose à l'écran, modifier un emplacement dans la mémoire, etc...

On a vu plus haut la commande pour afficher du texte à l'écran. On peut également utiliser cette commande pour afficher la valeur d'une variable. Testez le code suivant (à l'intérieur de la fonction main) :

```
int n=10;
printf("La variable n est égale à %d\n",n);
```

Les caractères `%d` indiquent que l'on veut afficher un nombre entier à cet emplacement. La variable contenant le nombre à afficher est indiquée en argument après la chaîne de caractères. Pour afficher un nombre à virgule, on utilise les caractères `%f`. Si la chaîne de caractères doit afficher la valeur de plusieurs variables, on écrit `%d` (ou `%f`) à chacun des emplacements où l'on veut afficher les variables, et on mets celles-ci en argument dans l'ordre d'affichage.

Voici la syntaxe du test conditionnel :

```
if (<condition) {
  <ce qui est fait si la condition est réalisée>
}
else {
  <ce qui est fait si elle ne l'est pas>
}
```

La partie `else { ... }` est facultative. On peut également utiliser `else if ( ... ) { ... }` si on a plusieurs conditions à tester.

Les principaux opérateurs de tests sont `==` pour tester l'égalité *(Attention au fait qu'il y a deux signes « égal »)*, `<`, `>`, `<=`, `>=` pour les inégalités strictes ou larges, `!=` pour tester la différence. On peut utiliser les opérateurs booléens `&&` pour « et », `||` pour « ou » et `!` pour « non ».

Voici la syntaxe des boucles `while` et `for`.

```
while (<condition>) {
 <ce qui est réalisé tant que la condition est vraie>
}
```

Pour la boucle `for`, voici un exemple.

```
int i;
for (i = 0; i<=10; i++) {
 [...]
}
```

Le compteur `i` doit être déclaré avant de commencer la boucle. Dans les parenthèses suivant le `for`, on commence par donner la valeur initiale au compteur, puis la condition qu'il doit vérifier tout au long de la boucle, puis la manière dont il est incrémenté. La syntaxe `i++` est un raccourci pour `i = i+1`. On pourrait remplacer le `i++` par `i+=2` (qui est un raccourci pour `i = i+2`) si on veut que le compteur soit incrémenté de 2 à chaque pas. On pourrait aussi utiliser une syntaxe du type `(i = 10 ; i>=0 ; i--)` pour une boucle décroissante.

Exercice 2. Faire afficher les nombres de 1 à 10 à l'écran à l'aide d'une boucle.

4. Le hasard

Pour tester nos fonctions, on aura souvent besoin de tirer des nombres au hasard. Pour cela, il faut ***une seule fois dans le programme*** initialiser le générateur de nombres aléatoires. Cela se fait par la commande

```
srand(time(NULL));
```

Ensuite, pour tirer un nombre au hasard, on utilise la fonction `rand()`. Celle-ci renvoie un nombre pseudo-aléatoire entre 0 et... un nombre très grand. Pour renvoyer, par exemple, un nombre entre 0 et 99, on peut réduire le résultat modulo 100. L'opération « modulo » se fait à l'aide de l'opérateur `%`. On peut donc écrire

```
int n;
n = rand() % 100;
```

et la variable `n` contiendra un nombre (pseudo-)aléatoire compris entre 0 et 99.

5. Les tableaux

Voici comment déclarer un tableau d'entiers (en l'occurrence, un tableau appelé `tab` contenant 10 entiers) :

```
int tab[10];
```

On peut également initialiser le tableau avec une syntaxe de la forme `int tab[3] = {2,5,0}`.

Pour accéder au i-ème élément du tableau une fois celui-ci déclaré, on écrit `tab[i]`. *Attention : la numérotation des éléments du tableau commence à* 0 *!*

Une fois que le tableau est déclaré, *on ne peut pas connaître sa taille !* Pour cette raison, dans toutes les fonctions prenant en argument un tableau, on prendra également en argument la taille dudit tableau.