- Introduction à l'algorithmique et au langage C -

Hello World!

TD3

1re année ESIEA - Semestre 1

L. Beaudoin & R. Erra & A. Gademer & L. Avanthey

2016 - 2017

Avant propos

*Tout apprentissage d'un langage, que ce soit en informatique ou en électronique, commence par la mise en œuvre du célèbre **Hello World !**. Il s'agit du programme minimum qui vous offre deux possibilités : vérifier que tout marche et démarrer de cette base pour y intégrer des actions plus complexes. La démarche de recopier et lancer le Hello World ! doit donc devenir automatique. Comme il s'agit de nos premiers pas en langage C, c'est ce par quoi nous allons commencer.*

*Pour pouvoir aller plus loin, nous introduisons directement un autre élément fondamental de l'algorithmique : les **variables**. En effet, comme nous l'avons vu, il nous faut pouvoir mémoriser des informations et les variables sont l'un de ces outils prévus à cet effet. Nous les appelons des structures de données.*

*Nous verrons également rapidement comment définir des **constantes** en langage C, c'est-à-dire utiliser des données qui ne changeront pas durant toute l'exécution du programme.*

1 Hello World !

1.1 Le plus petit programme C du monde !

Voici le plus petit programme minimaliste en C.

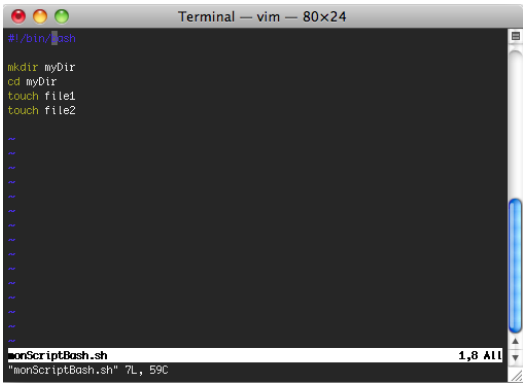
```
int main() {

   return 0;

}
```

Remarques sur la syntaxe
Le bloc `main` encadre tout l'algorithme et sa présence est obligatoire dans tous les programmes.
Instructions : elles se terminent toutes obligatoirement par un point-virgule.
Blocs : ils commencent par une accolade ouvrante et se termine par une accolade fermante. Ici aussi, nous indentons le contenu des blocs afin de clarifier la lecture.

Nous allons écrire ce programme. Nous avons vu que les programmes en langage C sont écrits dans des fichiers textes qui possèdent l'extension `.c`, nous allons donc avoir besoin d'un **éditeur de texte**.

1.2 Outils de travail

Nous utiliserons l'éditeur de texte `Vim`. C'est un éditeur de texte très pratique qui se lance dans le terminal. Nous l'utilisons très souvent pour éditer et modifier des fichiers ou lorsque nous n'avons pas de serveur graphique.



L'éditeur de texte `Vim`.

Attention !
Dans les années supérieures, vous découvrirez ce que nous nommons des Environnements de Développement Intégré (IDE pour Integrated Development Environment) comme **Netbeans**, **Éclipse**, **CodeBlocks** ou **Xcode**.

Ces environnements de travail permettent de travailler sur de très gros projets, de manière collaborative et avec des outils très puissants de manipulation de code.

Leur inconvénient majeur est qu'ils nécessitent une pléthore de fichiers auxiliaires (fichiers de projet), qui sont totalement superflus pour le moment et risque de vous embrouiller. Par ailleurs, en masquant les étapes de compilation derrière un clic bouton, **ils vous empêchent d'apprendre correctement votre cours**.

Pour bien travailler
`Microsoft Windows` n'est pas un environnement de travail propice pour apprendre la programmation. En effet, il n'intègre pas par défaut les outils de compilation et nécessite l'usage des environnements de développements pré-cités.

De plus, le comportement de vos programmes peut différer selon le système d'exploitation et le compilateur installé, ce qui pose des problèmes de compatibilité lors des rendus. Si `Mac OS X`, en tant que système UNIX pose moins de problèmes que `Microsoft Windows`, il n'est pas non plus totalement compatible avec `GNU/Linux`. **Vous ferez donc toujours attention à tester vos programmes sur des machines de l'École avant de les soumettre à vos professeurs !**

Pour travailler chez vous « comme à l'école »
Nous avons mis à votre disposition une machine virtuelle Ubuntu qui reprend les mêmes paramètres que les machine de l'école. Cela vous permet de « faire tourner » un OS `GNU/Linux` dans une fenêtre, que vous soyez sous `Microsoft Windows`, `Mac OS-X` ou `GNU/Linux`. Pour savoir l'installer et l'utiliser, consultez la fiche annexe que nous mettons à votre disposition.

Vous pouvez le lancer dans le terminal en lui indiquant directement le nom du ficher à éditer et si ce dernier n'existe pas, il sera automatiquement créé :

```
vim monFichier
```

EXERCICE 1

En vous inspirant de la commande ci-dessus, ouvrez avec `Vim` le fichier `helloWorld.c`.

L'interface de `Vim`

L'interface de `Vim` est assez inhabituelle mais redoutablement efficace une fois prise en main. **Toutes les opérations s'effectuent au clavier.** En particulier, **oubliez les menus** « Fichier », « Édition », etc., qui appartiennent au `Terminal` et non à `Vim`.

Voici les **commandes** essentielles pour débuter :

- Pour pouvoir écrire à l'intérieur du fichier, il faut passer en mode *Insertion* en appuyant sur la touche `I` (le mode s'affiche en bas à gauche de l'écran et vous pouvez écrire ce que vous voulez).
- Pour enregistrer votre fichier, il faut revenir en mode normal en appuyant sur la touche `echap` (le mot « Insertion » en bas à gauche disparaît alors, vous ne pouvez plus éditer le fichier), puis entrer la commande suivante : « `:w` ».
- Pour quitter il faut toujours être en mode normal et entrer la commande suivante : « `:q` ».
- Vous pouvez quitter et enregistrer en une fois en entrant dans le mode normal la commande suivante : « `:wq` ».

Fiche Annexe Vim

Vous trouverez sur la fiche annexe Vim de nombreuses autres commandes qui vous seront bien utiles, comme le copier-coller & co !

EXERCICE 2

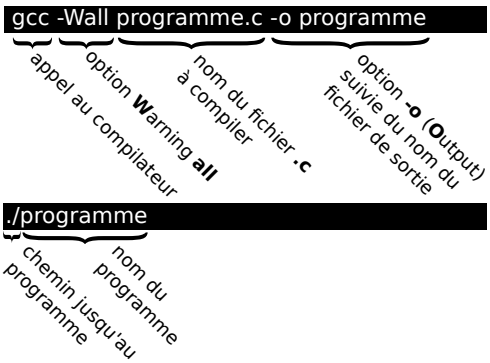
Recopiez le programme minimum dans votre fichier, sauvegardez puis quittez l'éditeur.

EXERCICE 3

À l'aide du terminal, réalisez maintenant ces différentes étapes :

- Compilez le programme `helloWorld.c` pour obtenir l'exécutable `a.out`
- Compilez le programme `helloWorld.c` pour obtenir l'exécutable `helloWorld`
- Exécutez `helloWorld`

Rappel pour compiler et exécuter un programme :



QUESTION 1


Que se passe-t-il lorsque vous exécutez ? Est-ce normal ?

Bonnes habitudes

Il est important de commenter vos codes afin de décrire ce qu'il s'y passe. En C nous utilisons soit :

```
/* My Comment */
```

pour un commentaire sur une ou plusieurs lignes. Il est aussi possible d'utiliser pour une seule ligne :

```
// My comment
```

1.3 Petit programme cherche à communiquer

Il ne se passe pas grand chose, mais rien de plus normal. Si nous ne faisons rien faire au programme, quand nous l'exécutons il ne fait donc rien. Mieux : il nous obéit au doigt et à l'oeil. Mais du coup ce n'est pas un vrai *Hello World !* que nous avons, car bien que nous n'ayons pas d'erreur, nous ne savons pas si tout marche bien. Pour qu'il puisse communiquer avec nous, nous voulons pouvoir afficher un message. La fonction qui permet cela est la fonction `printf` et il est important que vous puissiez rapidement la maîtriser, car elle vous servira en permanence.

1.3.1 Afficher du texte : principe

La fonction `printf` nous permet, entre autres, d'afficher du texte statique. Nous utilisons pour cela le modèle suivant (notez que les guillemets définissent une chaîne de caractères) :

```
printf("Our static message ");
```

Suivi du résultat à l'écran :

```
Our static message user@computer:~$
```

On notera que sans retour à la ligne final, l'invite de commande se retrouve directement à la suite de notre texte.

Nous obtiendrons le même message de cette manière :

```
printf("Our ");
printf("static message ");
```

Bien que nous ayons écrit le message en plusieurs instructions, le résultat est le même que si nous n'en avions qu'une. Si nous affichons toutes les informations dont nous avons besoin à la suite, tout cela risque de devenir très vite illisible. Pour éviter ce cas extrême, nous pouvons utiliser deux options de mise en forme : le retour à la ligne (`\n`) et la tabulation (`\t`).

QUESTION 2


D'après ces informations, écrivez dans le cadre ci-dessous ce que nous aurions nous à l'écran avec les instructions suivantes.

```
printf("Hello\n");
printf("\tMy name is HAL\n");
```

QUESTION 3

 Et avec celle-ci ?

```
printf("Hello\n\tMy name is HAL\n");
```

Question bonus : À quel livre (saga adaptée en partie au cinéma) faisons nous référence dans cet exemple ?

QUESTION 4

 Allez, un petit peu de pratique pour fixer les esprits avant de continuer. Écrivez **sur papier** les instructions permettant d'afficher à l'écran ce qui suit :

- Sur la première ligne `Science is what we understand well enough to explain to a computer.`
- Sur la seconde ligne `Art is everything else we do.`
- Et sur la troisième, précédée d'une tabulation : `Donald Knuth`[1] suivi d'un dernier retour à la ligne.

1.3.2 Afficher du texte : programme

Comme nous l'avons indiqué, `printf` est une fonction qui permet d'écrire un flux de sortie. Elle est décrite dans une bibliothèque que nous devons ajouter au programme pour pouvoir continuer. Il s'agit de la bibliothèque `stdio` (*STandarD Input/Output library*) qui gère les flux d'entrée et sortie standards dont nous avons parlé dans le TD « Initiation au Shell ».

Nous inclurons désormais les entêtes (header, possédant l'extension `.h`) **de la bibliothèque au début de notre programme** :

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

1.4 Bonjour Monde !

EXERCICE 4

 Complétez ce qui vous manquait dans votre programme `helloWorld.c` pour avoir un vrai *Hello World !*. Compilez et exécutez. Vérifiez que le programme vous écrit bien le message désiré.

Nous nous référerons au code qui suit comme le `helloWorld.c` tout au long de l'année.

```
#include <stdio.h>

int main() {

   printf("Hello World!\n");

   return 0;

}
```

```
Hello World!
```

EXERCICE 5

 Ajoutez les différentes instructions que nous avons vues à la section 1.3.1 (avec Hal et Knuth). Compilez et vérifier vos réponses théoriques.

1. Et si vous ne savez pas qui est Donald Knuth, vous avez gagné le droit d'aller lire sa bibliographie ;-)

2 Les variables

Cette première étape de communication avec le programme effectuée, nous allons vouloir aller un peu plus loin. Comme nous l'avons vu dans les précédents TD, il nous faut pouvoir stocker de l'information pour pouvoir la manipuler ensuite. Pour cela, nous utilisons ce que nous appelons des « variables ».

2.1 Comment ça marche ?

2.1.1 Déclaration et réservation mémoire

Les variables sont stockées dans la mémoire vive de l'ordinateur [2]. Cette dernière est constituée d'espaces mémoires qui peuvent être utilisés par les programmes. Une variable, par son nom, réserve un espace mémoire qui lui est dédié (Fig N°1).

Mais un ordinateur ne stocke et ne traite que des 0 et des 1. Un texte et un nombre sont donc tous les deux stockés en mémoire sous forme binaire [3]. Ce qui nous permet de les distinguer, c'est le type que nous donnons à la variable qui les contient (Fig N°1). Les entiers sont catégorisés par le type `int`, les flottants [4] par le type `double` et les caractères par le type `char`.

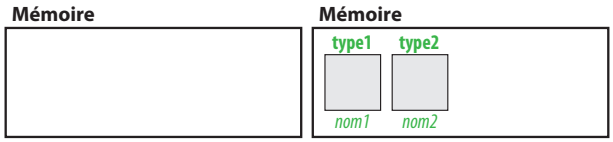


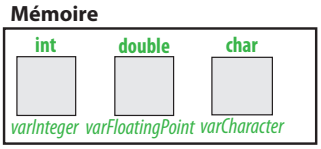
Figure 1 – ***Représentation de la mémoire (gauche) et réservation d'espace (droite)***

Le code qui suit vous montre comment déclarer des variables en langage C. Vous trouverez à côté la représentation en mémoire de ces actions.

EXERCICE 6

Rajoutez ces lignes dans votre programme `helloWorld.c`.

```c
int varInteger;
double varFloatingPoint;
char varCharacter;
```



Fondamental

Le nom d'une variable doit toujours être représentatif, clair et compréhensible. Les noms comme `a`, `b` ou `c` sont à proscrire (sauf si les variables représentent des éléments mathématiques). Ces noms ne doivent être ni trop longs (`name-of-my-variable`) ni trop courts (`vn`), juste ce qu'il faut : `varName` ou `variableName`.

Ils doivent être en anglais et commencent obligatoirement par une minuscule.

2. Couramment appelée RAM (*Random Access Memory*).
3. Nous verrons dans le TD « Bases & représentation des données » comment l'ordinateur traduit les concepts humains en séries de 0 et de 1.
4. Les flottants ou *nombres à virgule flottante* représentent des valeurs non entières, approximations de nombres réels.

QUESTION 5

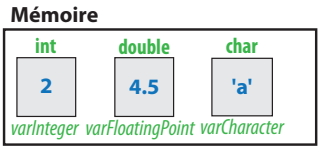
Appliquons ce que nous venons de voir. Comment faire pour :
- Déclarer une variable devant contenir un nombre d'élèves ?
- Déclarer une variable devant contenir le prix d'un livre ?
- Déclarer une variable devant contenir le numéro d'un mois ?
- Déclarer une variable devant contenir la première lettre d'un mois ?

2.1.2 Affectation et état mémoire

Maintenant il nous reste à placer une donnée dans l'espace mémoire que nous avons réservé et étiqueté, c'est-à-dire que nous allons donner une valeur à notre variable. C'est ce que nous appelons une **affectation**.

Voici comment faire en langage C et la représentation de la mémoire associée.

```
varInteger = 2;
varFloatingPoint = 4.5;
varCharacter = 'a';
```



EXERCICE 7

Rajoutez ces trois affectations dans votre programme `helloWorld.c`.

Il va sans dire qu'il est impossible de donner une valeur à une variable sans l'avoir préalablement déclarée (souvenez vous, la lecture d'un programme est séquentielle). En effet, dans la figure n°1 page 6, nous voyons bien sur le schéma de gauche que nous n'aurions nulle part où mettre notre valeur (pas de carré gris).

2.2 Affichage de variables

Pour être sûr que tout cela marche bien, nous aimerions à nouveau que le programme nous le dise lorsqu'il s'exécute. Nous allons réutiliser la fonction `printf`, cette fois-ci un peu différemment, car nous ne voulons plus afficher un texte statique, mais le contenu d'une variable.

Alors, nous savons que nous pouvons accéder au contenu en appelant la variable par son nom. Maintenant il faut savoir comment le donner à la fonction `printf`. C'est assez simple. D'une part, nous allons lui fournir le nom de la variable qui nous intéresse, et d'autre part, dans la chaîne de caractère (délimitée par les guillemets, souvenez-vous) nous allons lui indiquer la position où nous voulons la mettre et sous quelle forme.

Si nous reprenons nos précédentes variables que nous avons déclarées et initialisées, cela donne par exemple :

```
printf("%d\n", varInteger);
```

Remarquez le `%d`, il s'agit d'un **descripteur de type**. Les lettres qui suivent le caractère `%` servent à spécifier le type de la variable `varInteger`. Le `d` signifie donc type entier en base dix (`int`). Pour un flottant (`double`), c'est `lf`. Et pour un caractère (`char`) c'est `c`.

```
printf("%lf\n", varFloatingPoint);
printf("%c\n", varCharacter);
```

Lors de l'affichage à l'écran les descripteurs sont remplacés par les contenus de chaque variable :

```
2
4.500000
a
```

Deux astuces de mise en forme :

— Nous pouvons ajouter du texte static autour du descripteur de type.

```
printf("The content of the variable named varInteger is: %d\n", varInteger);
```

```
The content of the variable named varInteger is: 2
```

— Nous pouvons afficher plusieurs variables dans une même instruction, il suffit de les lister. Attention à bien respecter l'ordre de la liste par rapport à l'ordre dans lequel vous avez placé les descripteurs de type.

```
printf("varInteger=%d, varFloatingPoint=%lf and varCharacter=%c\n", varInteger,
    varFloatingPoint, varCharacter);
```

```
varInteger=2, varFloatingPoint=4.500000 and varCharacter=a
```

2.3 Exercices

EXERCICE 8

Appliquons tout ce que nous venons de voir :

a) Déclarez la variable `anInteger` de type entier et affectez lui la valeur 12.

b) Déclarez la variable `aLetter` de type caractère et affectez lui la valeur `'b'`.

c) Déclarez la variable `aFloatingPoint` de type flottant et affectez lui la valeur 3.5.

d) Déclarez la variable `anotherInteger` de type entier et affectez lui la valeur 43.

e) Affichez le contenu de ces variables avec le format approprié et vérifiez ainsi ce que vous avez fait.

EXERCICE 9

Même nom, même type : déclarez à la suite une nouvelle variable nommée `anotherInteger` de type entier et affectez lui la valeur 5.

QUESTION 6

Que se passe-t-il lorsque vous compilez ? Avons-nous le droit de déclarer une variable de même type et de même nom qu'une variable déjà déclarée dans le programme ? Pourquoi ?

EXERCICE 10

Même nom, type différent : effacez la déclaration précédente et déclarez une nouvelle variable `anotherInteger` de type flottant et affectez lui la valeur 5.0.

QUESTION 7

Que se passe-t-il lorsque vous compilez ? Avons-nous le droit de faire cela ? Pourquoi ?

EXERCICE 11

Effacez la déclaration précédente pour que le code compile. Affectez maintenant à la variable `anInteger` le contenu de la variable `anotherInteger`. Puis affectez à la variable `anotherInteger` le contenu de la variable `anInteger`.

QUESTION 8

 Qu'est devenue l'ancienne valeur 12 de `anInteger` ? Existe-t-elle encore ? Faites un dessin pour montrer ce qui s'est passé en mémoire.

2.4 Algorithme : Échange de variables

Allez, un dernier petit exercice pour la route, afin de vérifier que vous avez bien compris le lien « variable et mémoire ». Nous avons deux variables de type entier `var1` et `var2`.

QUESTION 9

 Nous voulons échanger le contenu de ces deux variables sans les perdre, quelque soient leurs valeurs d'origine. Comment faire ? Écrivez les instructions en pseudo-code ou en langage C.

EXERCICE 12

 Testez votre idée dans le programme `helloWorld.c` (n'oubliez pas d'afficher le contenu des variables avant puis après l'échange).

3 La notion de constantes

Ce que nous entendons par constantes, ce sont des **valeurs numériques** qui ne seront **pas amenées à évoluer**. Ce ne sont donc pas des valeurs que nous plaçons dans des variables.

Prenons π par exemple, et disons que vous ayez besoin d'une précision de 30 décimales. Pour peu que vous ayez à l'utiliser plusieurs fois dans le programme à différents endroits, vous vous voyez écrire (même en copier-coller) dix fois 3.14159265358979323846264338379 ? Et si jamais vous décidez après coup finalement que 20 décimales seulement vous suffisent, même si vous disposez de l'option rechercher-remplacer, il vous faut alors tout modifier.

C'est pour éviter ce genre de désagréments, qu'il est possible de donner un nom à nos constantes. Elles sont alors définies à un seul endroit, et on utilise leurs noms partout où nous en avons besoin.

En C, nous l'écrivons ainsi (à placer tout en haut du programme, sous les `#include`) :

```
#define PI 3.14159265358979323846264338379
```

Attention

La définition de constante n'est pas une attribution, il n'y a donc aucune raison d'utiliser l'opérateur `=`. Il n'y a pas non plus de point-virgule à la fin.

Le nom de notre constante ici est `PI`. Conventionnellement (pour mieux les reconnaître en tant que constantes) nous écrivons ces noms toujours en majuscules. Le compilateur, au tout début de son action, remplacera toutes les occurrences du mot `PI` qu'il trouvera dans tout le programme par la valeur que nous avons mis immédiatement derrière lui dans le `#define`, soit 3.14159265358979323846264338379.

Notez bien

Cela fait deux fois maintenant que nous croisons des instructions qui commencent par un dièse (#) et que nous plaçons tout en haut de nos programme : `#include`, `#define`. Nous appelons ces instructions des **commandes préprocesseur**, car elles sont appelées par le compilateur avant l'étape de compilation.

Pour voir l'effet des commandes préprocesseurs vous pouvez utiliser la commande :

```
gcc -E programme.c -o programme.txt
```

(L'option `-E` arrête la compilation après le pré-processeur.)

QUESTION 10


Pour être sûr d'avoir bien compris, complétez le tableau suivant :

TABLE 1: Constantes

Nom	Désignation	Valeur	Instruction correspondante en C
Nombre d'or	ϕ	$1,61803398$	
Constante de Neper	e	$2,71828182$	
Nombre Pi	π	$3,14159265$	
Vitesse de la lumière	c	299792458	
Accélération de pesanteur	g_0	$9,80665$	

Mémo

« Tout commence par un Hello World ! »
« Structure de données : les variables. »
« Une variable est caractérisée par un nom et un type ; elle contient une donnée. »
*« Trois types : **int** pour les entiers, **double** pour les flottants, **char** pour les caractères. »*