TP n°2 – Introduction au C

1. Compilation

- **Q1.** Ouvrir le fichier `tp02.c` avec un éditeur de code. Dans un terminal, compiler avec `gcc -o tp02.out tp02.c` .

Ici vous devriez avoir une erreur similaire à "référence indéfinie vers « main »". Un fichier en C doit toujours comporter une fonction nommée `main`.

Le `main` est la porte d'entrée du fichier : lorsqu'on exécute l'exécutable, cela fait ce qui est écrit dans le `main`.

À partir de maintenant vous écrirez toute ligne de code dans le `main`, sauf si c'est la définition d'une fonction (comme la fonction `f` donnée en exemple)

- **Q2.** Ajouter à la fin du fichier :

```c
int main(){
   int x = 4*2;
   printf("%d\n",x);
}
```

- **Q3.** D'après vous, que fait ce code ? Vérifiez en compilant à nouveau et en exécutant avec la commande `./tp02.out`

Il y a trois points remarquables dans le code (observer les exemples donnés) :

- Lorsqu'on définit une fonction en C, on écrit son type de retour, son nom, ses arguments entre parenthèses avec leurs types et le corps de la fonction entre { et }. Il doit y avoir un `;` après chaque instruction dans le corps d'une fonction.
  L'instruction `return` permet de renvoyer une valeur. Elle arrête la fonction. Elle est obligatoire pour renvoyer une valeur, sauf dans `main`.
  Par exemple la fonction `f` est de type de retour `int` et ses arguments sont `x` et `y`, tout deux des `int` (entiers).
  Pour appeler la fonction, on écrit son nom suivi des arguments entre parenthèses. Par exemple `f(3,4)`.
- Lorsqu'on définit une variable en C, on écrit son type, son nom et sa valeur après le signe =. Par exemple `int a = 3`.
- La seule manière de savoir ce que du code C fait est de lui faire afficher en utilisant `printf`.
  La fonction `printf` prend en argument une chaine de formattage et les donner à afficher. Son type de retour est `void`, c'est à dire rien.
  La chaine de formattage est une chaine de caractères qui contient des indications de placement indiquées par des % suivis de lettres indiquant le type de la donnée (d pour un entier, c pour un caractère, s pour une chaine de caractères, f pour un flottant). Il doit y avoir autant de % que de données à afficher.
  Par exemple `printf("un entier : %d, un mot : %s", 6, "patate")` affiche à l'écran `un entier : 6, un mot : patate`.

- **Q4.** Écrire une fonction `int g(int x)` qui renvoie x multiplié par 5.
- **Q5.** Dans `main`, afficher le résultat de `g(4)`.

Erreurs et avertissements

- **Q6.** Créer une fonction `h` de type de retour `void`, sans aucun arguments et de contenu `return 0;`. Appeler `h` dans le `main` en écrivant `h();` (pas de besoin de `printf`)
  Un avertissement apparait pour vous indiquer un problème. Cependant vous pouvez exécuter sans problème.

Les avertissements de gcc sont souvent très utiles pour éviter des erreurs à l'exécution. Parfois ils ne produisent pas d'erreurs mais montrent que votre code est mal écrit. C'est une bonne pratique de toujours les résoudre, même quand ce qu'ils indiquent n'est pas important.

2. Variables et types

Une variable en C se déclare (on lui donne un nom et on trouve de la place en mémoire pour la mettre) puis s'affecte (on remplit la case). Entre la déclaration et l'affectation la case contient une valeur mal définie. Des exemples se trouvent dans le fichier du TP.

- **Q7.** Déclarez un variable `c` de type `char` (caractère) et une variable `d` de type `double` (grand flottant/réel).
- **Q8.** Essayez d'afficher `d` et `c`. (il est très possible que vous tombiez sur 0.0 et un caractère invisible, mais ça ne veut pas dire que ça sera toujours le cas)
- **Q9.** Déclarez un booléen (le type s'appelle `bool`) `b` (une variable qui est soit VRAI soit FAUX).

- **Q10.** Affectez `b` à `true`, `c` à 'j' (avec les guillemets simples) et `d` à 3.6 (on utilise un point plutot qu'une virgule dans les nombres à virgule).

Les opérateurs arithmétiques en C sont les suivants : +, -, *, /, %.

- **Q11.** Testez sur diverses variables, entières (int) et flottantes (double). Est il possible de les utiliser sur deux variables de types différents ?

Même si 0 et 1 peuvent représenter des booléens, le programme de MP2I précise clairement qu'on ne doit pas les utiliser pour cela et qu'il faut utiliser `true` et `false`.

Les opérateurs logiques booléens sont && (et), || (ou) et ! (non).

- **Q12.** Testez (il sera affiché 1 pour `true` et 0 pour `false`) :

```
bool b1 = true;
bool b2 = false;
bool b3 = b1 && b2;
bool b4 = (b1 || b2) && !b3;
printf("b3 = %d, b4 = %d", b3, b4);
```

Les opérateurs de comparaison sont les suivants : <, <=, >, >=, == (test d'égalité), != (test de non égalité).

- **Q13.** Testez les sur plusieurs valeurs de types différents. Peut-on comparer deux valeurs de types différents ? Quel est le type renvoyé ?

Il est possible de réaliser des affectations composées, en plaçant un opérateur arithmétique devant le '='.

- **Q14.** Testez en affichant les valeurs de a et b :

```
int a = 1, b = 3;
a += 5;
b /= 2;
```

En C, les variables normales peuvent être modifiées. Si on veut une variable dont on ne peut pas modifier la valeur, il faut déclarer une constante avec le mot-clef `const`.

- **Q15.** Testez :

```
const float PI = 3.14;
PI = 3.145;
```

C'est une très bonne pratique d'utiliser des noms en majuscule pour les constantes, mais ce n'est pas obligatoire.

3. Structures de contrôle

Les accolades définissent un "bloc" d'instructions. Les blocs sont utilisés pour définir le corps des boucles et des conditionnelles. Les variables définies à l'intérieur du bloc ne sont pas définies en dehors du bloc, elles sont locales.

Les conditionnelles s'écrivent ainsi :

```
if (x>0) {...} else {...}
```

Si on veut faire plusieurs conditionelles imbriquées :

```
if (x>0) {...}
else if (y=3) {...}
else {...}
```

et ainsi de suite.

- **Q16.** Écrire une fonction `plusque3` qui prend en paramètre un entier et affiche une phrase indiquant si cet entier est plus grand que 3 ou non. On affichera une phrase supplémentaire si l'entier est nul.

Les boucles for s'écrivent ainsi :

```
for (int i=0; i<n; i+=1){
...
}
```

Cette syntaxe correspond à la boucle "Pour i allant de 0 à $n-1$". La variable i est définie dans la boucle for mais pas en dehors.

Le code suivant calcule la somme $\sum_{i=1}^{n} 2*i$, pour n donné plus haut dans le code :

```
int s = 0;
for (int i=0; i<n; i+=1){
  s=s+2*i;
}
```

- **Q17.** Écrire une fonction qui permet de calculer la somme $\sum_{i=1}^{n} i^2$, pour n donné.

Les boucles while s'écrivent ainsi :

```
while (condition){
...
}
```

En C il existe des moyens de sortir d'une boucle for ou while avant sa fin en utilisant :

- `return` qui termine la fonction par la même occasion.
- `continue` qui permet de passer à l'itération suivante sans terminer l'itération actuelle.
- `break` qui interrompt la boucle.

- **Q18.** Testez :

```
for (int i = 0; i < 4; i += 1) {
  if (i == 2) {
      break;
  }
  printf("%d\n", i);
}
```

- **Q19.** Testez :

```
for (int i = 0; i < 4; i += 1) {
  if (i == 2) {
      continue;
  }
  printf("%d\n", i);
}
```

- **Q20.** Ré-écrivez ce dernier code avec une boucle while

4. Exercices

- **Q21.** Écrire une fonction `int factorielle(int n)` qui calcule $n!$. Calculer la factorielle des entiers 0, 1, 10, 20, 100, 10000. Que remarquez-vous ?
- **Q22.** Écrire des fonctions calculant : $G_n = \sum_{k=0}^{n} k^3$, $H_n = \sum_{k=1}^{n} \frac{1}{n}$.
- **Q23.** Écrire une fonction qui affiche tous les entiers entre 100 et 0 (compte à rebours) dans l'ordre décroissant.
- **Q24.** Écrire des fonctions qui permettent d'afficher des triangles à n lignes de la forme suivante (exemples donnés pour $n = 6$) :

```
*          *          *            *          *
**         --         **          ***        * *
***        ***        * *        *****      *   *
****       ----       *  *      *******    *     *
*****      *****      *   *    *********  *       *
******     ------     ******  ***********  ***********
```

5. Fonctions récursives

En C les fonctions peuvent être récursives, c'est à dire s'appeler elles-mêmes. Les fonctions récursives sont souvent utilisées pour calculer des relations de récurrence.

Par exemple si la suite $(u_n)_{n\in\mathbb{N}}$ est définie par $\forall n \in \mathbb{N}^*$, $u_{n+1} = u_n * 3 + 6$ et $u_0 = 1$, alors on peut calculer u_n pour n quelconque par :

```
int u_n(int n){
   if(n=0){return 1;}
   else{return 3*u_n(n-1)+6;}
}
```

On a fait appel au calcul `u_n(n-1)` dans la fonction `u_n` elle-même.

- **Q25.** Écrire une fonction qui calcule la suite définie par $u_{n+2} = 3 + u_{n+1} + 4 * u_n$ et $u_0 = 4$ et $u_1 = 7$.
- **Q26.** Écrire une fonction récursive qui calcule la factorielle d'un nombre n donné en entrée.