TP2 : Fonctions et boucles

Guillaume Rousseau
MP2I Lycée Pierre de Fermat
guillaume.rousseau@ens-lyon.fr

14 septembre 2022

Consignes

Vous devez déposer votre rendu sur le cahier de prépa de classe, avant le dimanche 18/09 à 22h00.

A partir de ce TP, mettez toutes vos réponses écrites dans un seul fichier texte. Appelez-le "reponses.txt" (sans accent sur le e) et mettez le dans le dossier du TP2.

Dans ce TP nous allons reprendre les éléments vus dans le dernier cours : les fonctions, les boucles for, les boucles while. Vous trouverez une fiche récapitulative de ce que l'on a vu jusqu'à présent dans les documents à télécharger du cahier de prépa

Programmes sans fin

Maintenant que l'on peut écrire des boucles et des fonctions récursives, nos programmes peuvent potentiellement boucler infiniment. Lorsque vous lancez un programme dans le terminal, vous pouvez l'arrêter de force avec Ctrl+C. Il faudra parfois le faire plusieurs fois pour que le programme réponde.

A Le type bool

On étudie de plus près les conditions des ***if-then*** et des ***for***.

Exercice 1. Écrivez un programme, utilisant un if-else dont la condition est simplement 0, puis un dont la condition est 1, puis un dont la condition est -5.68. Rajoutez de l'affichage dans les différentes branches. Comment se comporte ce programme ? Quelles conditions sont réalisées ?

En réalité, les conditions des if-then et des for sont des valeurs au même titre que les entiers et les flottants :

```
1 int main(){
2   int x = 0;
3   int y = 2;
4   int b = (x==y) || (x > y*5) ; // vaut 0
5   int b = (x < y); // vaut 1
6 }
```

Plus précisément, les opérateurs de comparaison (==, ! =, <, etc...) se comportent comme des fonctions, qui renvoient un entier. La valeur 0 correspond à une condition qui n'est ***pas***

vérifiée. Ainsi, lorsqu'un programme teste une condition dans un if ou dans une boucle for, il vérifie en réalité si la condition vaut 0 ou n'importe quelle autre valeur.

Donner le type int aux conditions est un peu contre-intuitif. On introduit donc un nouveau type : le type ***bool***. Ce terme vient du nom de Georges Boole, un mathématicien et logicien anglais. Les ***booléens*** sont au nombre de 2 : ***true*** et ***false***. Les opérateurs logiques && (ET), || (OU) et ! (NON) sont trois opérations sur les booléens. On parle ***d'algèbre de Boole*** (on en reparlera au cours de l'année quand vous aurez fait un peu plus de maths).

Pour utiliser le type bool, et les valeurs true et false, il faut utiliser une nouvelle librairie : ***stdbool.h*** !

Exercice 2.

1. Écrivez un programme créant deux variables booléennes a et b valant true et false respectivement, puis calculant et affichant $a, b, a\&\&b$, $a||b$ et $!a$ (avec %d). N'oubliez pas d'inclure la librairie stdbool.h !
2. Complétez les tables des valeurs de && et || :

x	y	$x\&\&y$
0	0	?
0	1	?
1	0	?
1	1	?

x	y	$x\|\|y$
0	0	?
0	1	?
1	0	?
1	1	?

B Fonctions

Lorsque l'on définit une fonction, on doit systématiquement écrire en commentaire ce qu'elle fait. Ce commentaire sert à expliquer aux personnes qui lisent le code à quoi sert la fonction, comment l'utiliser, et quelles restrictions éventuelles s'appliquent aux arguments. Par exemple :

```
1  #include <stdio.h>
2
3  /* Affiche x, n fois d'affilées. n doit être
4    positif ou nul */
5  void multi_affiche(float x, int n){
6    for (int i = 0; i < n; i++){
7      printf("%d\n", x);
8    }
9  }
10
11 /* Calcule et renvoie a^n modulo b. n doit être positif
12    ou nul, b doit être supérieur ou égal à 2. */
13 int puissance_mod(int a, int n, int b){
14   int result = 1;
15   for (int i = 0; i < n; i++){
16     result = (result * a) % b;
17   }
18   return result;
19 }
20
21 int main(){
22   // Ce programme fait .... et affiche ...
23   int x = puissance_mod(17, 6, 13);
24   multi_affiche(0.15, x);
25   return 0;
26 }
```

Il est important de noter que le commentaire expliquant une fonction fait toujours apparaître chacun des paramètres.

À partir de maintenant, lorsque vous écrivez une fonction, vous ***devez*** la commenter ainsi ! Vous devez également commenter vos programmes, soit juste avant le main soit au tout début du fichier, en expliquant brièvement ce que fait le programme, et ce même lorsque cela vous paraît évident.

Exercice 3. Créez un fichier mes_fonctions.c, et définissez les fonctions suivantes :

1. Une fonction qui, étant donné $a \in \mathbb{N}$ et $b \in \mathbb{N}^*$, détermine si b divise a.
2. Une fonction qui, étant donné x flottant non nul, affiche x et son inverse.
3. Une fonction qui étant donné x, y, z, t, calcule $3x + 5y - 6.25z + t$, l'affiche, et renvoie son carré.
4. La fonction main, qui devra tester les fonctions précédentes sur plusieurs valeurs.

C Assertions

La plupart des fonctions que l'on a écrit jusqu'à présent ont un défaut : Elles ne vérifient pas leurs entrées. Par exemple, la fonction factorielle ne vérifie pas si son entrée est bien dans $\mathbb{N}$. En conséquence, nos programmes peuvent boucler à l'infini alors que ce n'est pas voulu.

Exercice 4. 1. Recopiez et compilez le code suivant :

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

int factorielle(int n){
  /* Calcule la factorielle de n lorsque n est positive ou nulle.
   */

  if (n == 0){
    return 1;
  } else {
    return n * factorielle(n-1);
  }
}

int main(){
  int n;
  scanf("%d", &n);

  printf("%d\n", factorielle(n));

  return 0;
}
```

Exécutez le programme avec comme entrée $n = 5$, puis $n = -3$. Que se passe-t'il ?

2. Au début de la fonction factorielle (après les commentaires), rajoutez l'instruction suivante : `assert(n>=0);` et réessayez de tester $n = -3$ (n'oubliez pas de recompiler). Que se passe-t'il ?

La fonction `assert`, fournie par la librairie `<assert.h>`, sert à déterminer si une condition est remplie ou pas, et produit un message d'erreur lorsqu'elle ne l'est pas.

Lorsque l'on identifie des hypothèses nécessaires au bon fonctionnement d'une fonction, on doit inclure ces hypothèses dans le programme. A partir de maintenant, lorsque l'on écrira une fonction, on procèdera ainsi :

— Indiquer en commentaire le rôle de la fonction. On doit bien faire apparaître le nom de tous les paramètres de la fonction, et stipuler les hypothèses de la fonction

— Ajouter au début de la fonction des assertions afin de vérifier les hypothèses.

Les hypothèses nécessaires au bon fonctionnement d'une fonction s'appellent aussi des ***préconditions***.

Dans la suite de ce TP, Vous devez diviser vos programmes en fonctions, de façon à bien exhiber l'organisation logique derrière le programme. N'oubliez pas de commenter les fonctions et rajouter des assertions comme expliqué ci-dessus !

Exercice 5. Reprenez votre code de l'exercice 3, et copiez le dans un nouveau fichier C. Rajoutez à chaque fonction les assertions et commentaires nécessaires.

D Boucles for

Dans cette partie, vous devez utiliser des boucles for

Exercice 6.

1. Écrivez un programme demandant de rentrer un entier n et afffichant successivement tous les entiers entre 1 et n.
2. Écrivez un programme qui affiche un escalier : Il prend en entrée un entier n puis affiche n lignes. La i-ème ligne sera composée de $2i + 1$ fois le caractère "-" suivi du caractère "|". Vous devrez définir une fonction qui gère l'affichage d'une ligne.

E Boucles while

Dans cette partie, vous devez utiliser des boucles while

Exercice 7. Écrivez un programme qui demande à l'utilisateur de rentrer des nombres, jusqu'à ce que l'utilisateur rentre un nombre strictement négatif, et qui affiche alors la somme de tous les nombres positifs rentrés.

Exercice 8.

1. Écrivez un programme qui génère un nombre aléatoire entre 0 et 4999, et qui tente de le faire deviner à l'utilisateur, de la manière suivante :
   — L'utilisateur rentre un nombre
   — Le programme lui dit "Plus haut" ou "Plus bas" si le nombre n'est pas bon
   — Lorsque l'utilisateur a trouvé le bon nombre, le programme affiche "Gagné" et s'arête
2. Connaissez-vous une stratégie efficace pour ce jeu ?

F Fonctions récursives

Exercice 9. Dans cet exercice, vous n'avez pas le droit d'utiliser les boucles !

1. Écrivez un programme demandant un entier n à l'utilisateur et affichant les n premiers entiers.

2. Écrivez un programme demandant un entier et affichant chaque chiffre de cet entier sur une ligne différente. Si l'entier est négatif, le signe '-' doit apparaître au début sur une ligne à part. Par exemple, sur l'entrée −894, votre programme doit afficher :

```
-
8
9
4
```

3. Écrivez un programme demandant un entier et affichant chaque chiffre de cet entier sur une ligne différente, ***dans l'ordre croissant d'importance***. Si l'entier est négatif, le signe '-' doit apparaître au début sur une ligne à part. Par exemple, sur l'entrée −894, votre programme doit afficher :

```
-
4
9
8
```

Indice : il n'y a pas grand chose à changer du programme précédent pour obtenir le bon résultat.

4. Écrivez un programme qui, étant donné un entier n, affiche n fois "O" sur la même ligne, puis revient à la ligne.
5. En combinant les codes précédents, écrivez un programme qui, étant donné un entier n, affiche pour chacun de ses chiffres le nombre correspondant de "O" sur une ligne. Si l'entier est négatif, le signe '-' doit apparaître au début sur une ligne à part. Par exemple pour -894, le programme doit afficher :

```
-
OOOOOOOO
OOOOOOOOO
OOOO
```

Tout comme les variables, les fonctions en C peuvent être déclarées et définies à deux endroits différents du code. Par exemple :

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

// renvoie le double de n
int doubler(int n);

// renvoie le quadruple de n
int quadrupler(int n){
  return doubler(doubler(n));
}

int doubler(int n){
  return 2*n
}

int main(){
  int a = quadrupler(5);
  printf("%d\n", a);
  return 0;
}
```

Dans ce code, on a ***déclaré*** la fonction doubler, puis on l'a utilisé pour ***définir*** la fonction quadrupler, et enfin on a ***défini*** la fonction doubler. Notez que l'on doit préciser le type de retour à la déclaration ET à la définition
Déclarer une fonction sans la définir sert à signaler au compilateur que la fonction existe et que l'on va la définir à un moment. Si vous regardez le code source de gros logiciels écrits en C (VLC, OBS Studio par exemple), vous pourrez voir que les fichiers sources sont séparés en deux familles : les fichiers .h, où l'on déclare les fonctions, et les fichiers .c, où l'on définit les fonctions. Plus tard dans l'année, vous apprendrez à faire des projets avec plusieurs fichiers.

Exercice 10. Écrivez un programme contenant deux fonctions, ping et pong, toutes deux de signature void → void, telles que :

— ping() affiche "Ping" suivi d'un retour à la ligne, attend une seconde, et appelle pong()

— pong() affiche "Pong" suivi d'un retour à la ligne, attend une seconde et appelle ping()

Vous aurez besoin de la fonction usleep, dans la librairie <unistd.h>. Cette fonction prend en argument un entier n, et met en pause le programme pendant n microsecondes.

G Suite de syracuse

Soit $x \in \mathbb{N}$. La suite de Syracuse de x est une suite $(u_n)_{n\in\mathbb{N}}$ à valeurs dans $\mathbb{N}$ définie par :

$$\begin{aligned} u_0 &= x \\ u_{n+1} &= \begin{cases} \dfrac{u_n}{2} & \text{si } u_n \text{ est pair,} \\ 3u_n + 1 & \text{si } u_n \text{ est impair.} \end{cases} \end{aligned}$$

Voici la suite de Syracuse pour quelques valeurs de x :

x	u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7
1	1	4	2	1	4	2	1	4
5	5	16	8	4	2	1	4	2
6	6	3	10	5	16	8	4	2
2	2	1	4	2	1	4	2	1

On remarque que pour les 4 valeurs données, la suite de Syracuse devient cyclique à 4-2-1 éventuellement. Cette suite fait l'objet d'une conjecture très célèbre : la ***Conjecture de Syracuse*** :

Conjecture 1. Pour tout $x \in \mathbb{N}$, en notant $(u_n)_{n\in\mathbb{N}}$ la suite de Syracuse de x, il existe $k \in \mathbb{N}$ tel que $u_k = 1$.

Autrement dit, la conjecture de Syracuse dit que la suite de Syracuse de tout entier $x \in \mathbb{N}$ devient cyclique à 4-2-1 éventuellement. Cette conjecture est toujours ouverte aujourd'hui [1].

Définition 1. Étant donné $x \in \mathbb{N}$, et $(u_n)_{n\in\mathbb{N}}$ sa suite de Syracuse, le ***temps de vol*** de x est le plus petit $k \in \mathbb{N}$ tel que $u_k = 1$. Si la suite n'atteint jamais 1, le temps de vol de x est $+\infty$

Exercice 11. Pour cet exercice, créez un unique fichier C, dans lequel vous écrirez plusieurs fonctions.

1. Créez une fonction suivant qui, étant donné $x \in \mathbb{N}$, calcule le terme suivant dans la suite de Syracuse.
2. Créez une fonction syracuse qui, étant donné $x \in \mathbb{N}$ et $n \in \mathbb{N}$, calcule le n-ème de la suite de Syracuse. Utilisez la fonction précédente pour simplifier votre code. Combien vaut le n-ième terme de la suite de Syracuse de x dans les cas suivants :

   a) $x = 9$, $n = 6$
   b) $x = 77$, $n = 128$
   c) $x = 1023$, $n = 729$
   d) $x = 1234567$, $n = 52397$

1. Si vous trouvez une preuve n'hésitez pas à m'en faire part

3. Créez une fonction temps_de_vol qui, étant donné $x \in \mathbb{N}$, calcule son temps de vol. Quel est le temps de vol de x dans les cas suivants :

   a) $x = 1$
   b) $x = 26$
   c) $x = 27$
   d) $x = 28$
   e) $x = 77030$
   f) $x = 77031$

4. Écrivez une fonction plus_long_vol qui, étant donné un entier N, renvoie l'entier $x \in [\![1, N]\!]$ ayant le plus long temps de vol. Donnez le résultat de cette fonction, ainsi que le temps de vol correspondant, pour :

   a) $N = 10$
   b) $N = 100$
   c) $N = 1000$
   d) $N = 10000$
   e) $N = 100000$
   f) $N = 1000000$
   g) $N = 10000000$

5. Combien de temps prend votre programme environ pour la dernière valeur ? Quelles améliorations pouvez-vous proposer pour le rentre plus efficace (pas besoin de les implémenter) ?

H Exercice Libre

Exercice 12. Imaginez un programme qui utilise les notions suivantes : [2]

— fonctions
— boucles for
— boucles while
— assertions et commentaires de fonctions

Pensez bien à planifier votre programme, à réfléchir en amont aux fonctions dont vous aurez besoin. Commentez ce programme (et toutes les fonctions !) pour expliquer succintement ce qu'il fait. Si vous n'avez pas d'idées vous pouvez piocher dans la liste suivante :

— Un programme qui teste si un nombre est premier
— Un programme qui calcule les n premiers termes de la suite de fibonacci.
— Un programme qui affiche des lignes de "*" de longueurs qui varient avec le temps (sinusoïdal, linéaire, autre), créant une sorte de vague dans le terminal
— Un programme qui calcule la moyenne, le min et le max des nombres rentrés par l'utilisateur, au fur et à mesure qu'il les rentre.

Pensez à bien tester vos programmes avec plusieurs valeurs, dont des valeurs limites et des très grandes valeurs, afin de vérifier le bon fonctionnement de votre code.

2. Si le programme que vous avez rendu à l'exercice libre du TP1 répond presque aux attentes, vous pouvez vous contenter de le modifier pour rajouter ce qu'il manque.