- Introduction à l'algorithmique et au langage C -

Boucles & Embranchements

TP n°2

1re année ESIEA - Semestre 1

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

2016 - 2017

Avant propos

Nous avons vu dans le TP « Calculette » comment écrire des embranchements en langage C. Aujourd'hui, nous allons compléter ce savoir en apprenant à écrire dans ce langage les autres structures de contrôles que nous avons vues dans le TD « Déroulement d'un algorithme » : les boucles.

Nous approfondirons le tout par une série d'exercices qui emploient ces deux concepts.

1 Les boucles

Comme pour les embranchements, la structure n'est guère différente en langage C (à part quelques éléments de syntaxe).

Nous allons séparer en deux catégories les boucles. Les premières sont dites **événementielles** : nous **ne connaissons pas** le nombre de tours de boucle que nous allons faire, car l'arrêt de la boucle dépend d'un événement. Nous retrouvons dans cette catégorie la boucle TANT-QUE. En langage C, nous l'écrirons sous cette forme :

```
while (TEST) {
   // Actions to repeat
}
```

La boucle de type REPTER TANT-QUE appartient elle aussi à cette première catégorie. Elle s'écrit ainsi :

```
do {
   // Actions to repeat
} while (TEST);
```

Celles de la seconde catégorie, quant à elles, sont **itératives** : elles utilisent un compteur. Nous les employons quand **nous savons combien de fois** nous devons boucler. Il s'agit des boucles POUR.

Nous les écrivons de cette manière :

```
DECLARATION DU COMPTEUR
for (INITIALISATION ; TEST ; INCREMENTATION) {
   // Actions to repeat
}
```

Cette boucle mérite quelques commentaires. Tout d'abord l'instruction `for` attend **trois** expressions.

— La première concerne l'initialisation du compteur : c'est une affectation.
— La deuxième concerne le test de sortie de la boucle : la boucle s'arrête quand le test devient **faux**.
— La troisième concerne la modification du compteur : c'est généralement une incrémentation.

Attention

Notez bien que ces trois expressions sont séparées par des point-virgules et non des virgules !

2 Correcteur automatique

Pour ce TP, vous allez découvrir l'utilisation d'un outil qui vous permettra de soumettre et valider vos exercices : le correcteur automatique. Il est pour le moment hébergé à l'adresse : http://autocorrect.esiea.fr.

Vous pourrez vous identifier avec les deux informations suivantes :
— Login : le même que votre login esiea (celui qui se trouve dans votre mail : `tartenpion` pour `tartenpion@et.esiea.fr`, attention pas de `etd-p\` devant !).
— Mot de passe : `esiea2012`.

2.1 Commencez par changer votre mot de passe

— Cliquez sur `Paramètres de tartenpion`.
— Saisissez l'ancien mot de passe (`esiea2012`) puis deux fois votre nouveau mot de passe.
— Cliquez sur `Appliquer`.

2.2 Sessions

Pour ce TP, la session du correcteur automatique est dite « libre ». C'est-à-dire que vous pouvez soumettre vos exercices autant de fois que vous le souhaitez. Nous verrons plus tard d'autres types de sessions.

3 Série harmonique

Nous allons écrire un programme qui calcule la somme d'une série harmonique pour un rang donné.

Exemple de sortie attendue du programme pour un rang égale à 5 :

```
Rank of the Harmonic serie?
5
1/1 + 1/2 + 1/3 + 1/4 + 1/5 = 2.283333
```

EXERCICE 1

Créez un fichier `harmonic.c` et écrivez-y le code du `Hello World!`. Vous supprimerez le `printf` après avoir testé la compilation.

EXERCICE 2

Rajoutez les lignes pour demander à l'utilisateur d'entrer au clavier le rang de la série harmonique (avec la fonction `scanf`). N'oubliez pas d'affichez le message de la question comme indiqué dans la description de la sortie attendue du programme ci-dessus : « Rank of the Harmonic serie ? » avant d'attendre le nombre au clavier. Tant que le nombre n'est pas strictement positif, vous lui reposez la question (message + attente du nombre). Si `scanf` renvoi autre chose que 1, vous afficherez `"Input error\n"` puis vous quitterez le programme en retournant -1.

Autocorrect
Le correcteur automatique va comparer la sortie de votre programme avec une sortie de référence. Ainsi, il faut scrupuleusement que votre sortie ressemble (à l'espace près) à celle donnée en exemple. Sinon votre programme ne pourra pas être validé !

Somme d'une série harmonique
Pour rappel, voici comment calculer la somme d'une série harmonique :
$1/1 + 1/2 + 1/3 + ... + 1/rang$

EXERCICE 3

Commencez par affichez à l'écran les termes de la somme...

```
1/1 + 1/2 + 1/3 + 1/4 + 1/5 + 1/6
```

EXERCICE 4

Rajoutez ensuite les lignes pour calculer la somme de la série harmonique à partir du rang donné (comme indiqué au début du paragraphe).

EXERCICE 5

Enfin, affichez à l'écran le détail de la somme, suivi du résultat.

Autocorrect
Quand vous avez vérifié votre travail (en testant l'exemple présenté au début du paragraphe), soumettez-le au correcteur automatique !

4 La pyramide inversée

Dans cet exercice, nous vous proposons de réaliser une pyramide symétrique d'étoiles inversée.

Exemple de sortie attendue du programme pour une hauteur de lignes égale à 10 :

```
Height in lines of the pyramid?
10
******************
 ****************
  **************
   ************
    **********
     ********
      ******
       ****
        **
         *
```

EXERCICE 6

 Créez le fichier `pyramide.c` et écrivez-y le code du `Hello World!`. Vous supprimerez le `printf` après avoir testé la compilation.

EXERCICE 7

 Rajoutez les lignes pour demander à l'utilisateur d'entrer au clavier la hauteur de la pyramide. Tant que le nombre n'est pas strictement positif, vous lui reposez la question. Si `scanf` renvoi autre chose que 1, vous afficherez `"Input error\n"` puis vous quitterez le programme en retournant -1.

EXERCICE 8

 Rajoutez ensuite les lignes afin d'afficher à l'écran la pyramide inversée comme illustrée dans l'exemple de sortie. Pour cela vous commencerez par réfléchir sur le nombre d'espaces à gauche qu'il doit y avoir en fonction du numéro de ligne, puis le nombre d'étoiles.

Autocorrect

Quand vous avez vérifié votre travail, soumettez-le au correcteur automatique !

5 Stars Matrix Like

Nous vous proposons dans cet exercice de faire défiler sur la console des lignes d'étoiles aléatoirement remplies du bas vers le haut.

Exemple de sortie attendue par le programme pour un nombre de lignes égale à 10 :

```
Please, enter a number of lines
10
* * **** *** ** *** * *** ** * * ****** *** ** *
    *** * * * * * **** ** ***** ** ** * *** *
***** * ** * * * * *** * ***** ** * * ** * * *
 ** *** *** ** * **** ** * ** * * ** ** *** ** * **
**** * **** ***** *** **** *** ** * ** * * ** ***
* *** * * *** ** ** ** *** * * *** *
**** * ****** * ** *** ** * * ** ** * * *** *
*** * *** * * *** * * * ** * ****** **** * * *** *
  *** * * ** *** ** *** ** ** * *** ** * * * **
  * * * ** * ******* ** *** * *** ***** ** * ** **
```

EXERCICE 9

 Créez le fichier `starsMatrix.c` et écrivez-y le code du `Hello World!`. Vous supprimerez le `printf` après avoir testé la compilation.

EXERCICE 10

 Rajoutez les lignes pour demander à l'utilisateur d'entrer au clavier le nombre de ligne à afficher. Tant que le nombre n'est pas strictement positif, vous lui reposez la question. Si `scanf` renvoi autre chose que 1, vous afficherez `"Input error\n"` puis vous quitterez le programme en retournant -1.

EXERCICE 11

 Ajoutez les lignes pour boucler sur ce nombre de lignes.

EXERCICE 12

 Vous fixerez la largeur d'une ligne à 60. Ajoutez une nouvelle boucle pour boucler sur toutes les colonnes (60 en tout donc) de chaque ligne. Pour chaque colonne affichez aléatoirement le caractère `*` ou le caractère espace.

Aléatoire, comment faire ?

Nous utiliserons une fonction qui va nous générer des nombres pseudo-aléatoire. Ce ne sont pas des vrais nombres aléatoires car il est très difficile d'en créer (et impossible pour un ordinateur). De plus si nous les générons, nous suivons un algorithme et cela n'a rien d'aléatoire. Par contre nous sommes capables d'approcher les propriétés du hasard.

La fonction que nous utiliserons en langage C s'appelle `rand` (bibliothèque : `stdlib.h`). Si nous la mettons dans une boucle, l'instruction `rand()` nous fournira la même séquence de nombres pseudo-aléatoires à chaque exécution. Nous verrons un plus tard comment nous approcher encore plus de la notion d'« aléatoire ».

Nous arrivons donc à récupérer un nombre pseudo-aléatoire, mais ce dernier est un entier. Or nous avons besoin d'un cas binaire : soit nous affichons l'étoile, soit nous affichons une espace. Il nous faut donc remettre cet entier sur une plage de deux valeurs uniquement, par exemple entre 0 et 1. Pour cela, nous pouvons utiliser l'opérateur modulo. Voici un exemple :

```
int alea;

alea = rand()%2 // Alea is equal to 0 or to 1
```

Pour valider l'exercice sur le correcteur automatique, nous considérerons de manière arbitraire que nous affichons une étoile quand le nombre aléatoire vaut 0, et une espace quand il vaut 1.

EXERCICE 13

 Petite variante : sur nos machines actuelles, l'affichage des lignes est très rapide. Si vous souhaitez ralentir l'affichage des lignes et obtenir un effet d'animation, il faut ralentir l'exécution du programme. Vous pouvez utiliser pour cela la commande `usleep`. Pour l'utiliser vous devez ajouter la bibliothèque `unistd.h`. Ajoutez ensuite dans la boucle qui affiche les lignes l'instruction suivante : `usleep(100000)`.

Autocorrect

Quand vous avez vérifié votre travail, soumettez-le au correcteur automatique !