Boucles

Bruno Bouzy
1er septembre 2018

Ce document rassemble des exercices de C sur les instructions répétitives ou « boucles »: `for`, `do while` et `while`.

Exercice 1

Ecrire un programme `continue.c` qui affiche le menu suivant:

```
0: quitter
autre: continuer
```

On suppose que l'utilisateur tape un premier caractère différent de 'entrée', appelé caractère déterminant, puis 'entrée'. Si le caractère déterminant est `0`, le programme se termine. Sinon le programme continue et le menu est affiché à nouveau. On écrira trois fois ce programme: en utilisant le `while`, en utilisant le `do while`, et en utilisant le `for`.
Quels sont les avantages et inconvénients de ces trois instructions ?

Exercice 1b

On souhaite rendre le programme `continue.c` plus robuste. On va donc écrire un programme `continue2.c` améliorant le programme `continue.c`. Une itération du programme `continue2.c` correspond à l'affichage du menu, à la lecture de un ou plusieurs caractères au clavier, le dernier caractère tapé étant toujours 'entrée'.
Successivement, on supposera que l'utilisateur peut taper :
a) immédiatement 'entrée' auquel cas le programme continue (pas de caractère déterminant).
b) des espaces avant de taper le caractère déterminant.
c) des caractères quelconques entre le caractère déterminant et 'entrée'.
On utilisera des `do while`.

Exercice 2

Ecrire un programme `hellorepet.c` qui demande à l'utilisateur un nombre positif et affiche autant de fois `hello` sur une ligne de l'écran, en décalant le `hello` d'un espace vers la droite à chaque ligne:

```
entier positif ? 4
hello
 hello
  hello
   hello
```

On écrira trois fois ce programme: en utilisant le `while`, en utilisant le `do while`, et en utilisant le `for`.
Quels sont les avantages et inconvénients de ces trois instructions ?

Exercice 3

Ecrire un programme `notede0a20.c` demandant à l'utilisateur de taper une note entre 0 et 20 tant

que celui-ci ne tape pas un nombre entre 0 et 20, et le félicitant lorsqu'il a réussi. On écrira trois fois ce programme: en utilisant le `while`, en utilisant le `do while`, et en utilisant le `for`.

Exercice 4

Ecrire un programme `multiple3.c` demandant à l'utilisateur un nombre minimum, un nombre maximum et affichant les multiples de 3 compris entre ces deux nombres.

Exercice 5

Ecrire un programme `suite.c` demandant à l'utilisateur un nombre initial positif u_0 et affichant la suite telle que:

$u_{n+1} = 3u_n+1$ si u_n est impair,
$u_{n+1} = u_n/2$ si u_n est pair.

Exercice 6

1) Ecrire un programme `alea.c` affichant x nombres aléatoires compris entre 0 et 20. On utilisera la fonction `srand` pour initialiser la suite aléatoire avec une graine donnée par l'utilisateur. On utilisera `rand` pour tirer effectivement un nombre aléatoire. On fera un `#include <stdlib.h>` afin d'utiliser les fonctions `srand` et `rand`.
2) Etendre programme `alea.c` pour qu'il affiche la moyenne et la variance des nombres aléatoires.

Exercice 7a

Ecrire un programme `secret.c` demandant à l'utilisateur un nombre maximal `t`, tirant un nombre secret au hasard compris entre `0` et `t`, et demandant à l'utilisateur de deviner ce nombre. Si l'hypothèse de l'utilisateur est la bonne, le programme félicite l'utilisateur et se termine. Si l'hypothèse de l'utilisateur est trop grande, le programme affiche `trop grand`, si elle est trop petite, le programme affiche `trop petit`. Tant que l'hypothèse est mauvaise, le programme demande une nouvelle hypothèse.

Améliorer le programme pour permettre à l'utilisateur de rejouer s'il le souhaite.

Exercice 7b

Ecrire un programme `secretinverse.c` demandant à l'utilisateur un nombre maximal `t`, tapé au clavier, et un nombre secret au hasard, compris entre `0` et `t`, restant secret dans la tête de l'utilisateur (ne pas le taper au clavier, le programme ne le connaît pas a priori). Le programme doit deviner le nombre secret de l'utilisateur par dichotomie. Répétitivement, le programme affiche une hypothèse. Si l'hypothèse du programme est la bonne, l'utilisateur tape = et se termine. Si l'hypothèse du programme est trop grande, l'utilisateur tape >, si elle est trop petite, l'utilisateur tape <. Tant que l'hypothèse est mauvaise, le programme propose une nouvelle hypothèse.

Améliorer le programme pour permettre à l'utilisateur de recommencer.

TP

Exercice 1

Ecrire un programme `somme.c` affichant la somme de N nombres tapés au clavier.

Exercice 2

Ecrire un programme `posinega.c` calculant le nombre d'entiers positifs et négatifs dans une suite de N entiers tapés au clavier.

Exercice 3

Ecrire un programme `taux.c` calculant au bout de combien d'années une somme placée à un taux T aura doublé.

Exercice 4

Ecrire un programme `tablemulti.c` affichant la table de multiplication.

Exercice 5

Ecrire un programme `diviseurs.c` affichant les diviseurs d'un nombre entier N.

Exercice 6

Ecrire un programme `parfait.c` qui détermine si un nombre P tapé au clavier est parfait, c'est-à-dire égal à la somme de ses diviseurs. Par exemple, `6` est parfait car `1+2+3=6`, `28` est parfait car `1+2+4+7+14=28`.

Exercice 7

Ecrire un programme `ascii.c` affichant tous les caractères ASCII (de 0 à 255) par ligne de 32 et séparés par des espaces.

Exercice 8

Ecrire un programme `losange.c` affichant un losange de taille N lue au clavier. On utilisera le caractère `'*'` comme élément du losange.

Exercice 9

Ecrire un programme `factorielle.c` affichant la factorielle des nombres de 0 à N tant que le résultat est inférieur à `INT_MAX`. On fera un `#include <limits.h>`.

Exercice 10

Ecrire un programme `romain.c` affichant les puissances de 2 inférieures à 5000, en chiffres romains.

Exercice 11

Ecrire un programme `afficheFonction.c` affichant la fonction $y=f(x)=e^{-x}\sin(2\Pi x)$ avec `y` représenté horizontalement et `x` verticalement. Le programme affiche une ligne pour chaque valeur de `x` multiple de `0.05` compris entre `-1` et `+1`. Une ligne contient des `'*'` pour les valeurs de `y` inférieure ou égale à `f(x)` et des blancs sinon. On prendra `y` allant de `-1.5` à `+2.5` avec un pas de `0.05`. On pourra utiliser les fonctions `exp`, `sin`. On pourra définir une constante `PI` égale à `3.1415926`. On incluera `<math.h>`. On compilera avec l'option `-lm`.

Exercice 12

Ecrire un programme `premiers.c` affichant les nombres premiers inférieurs à un nombre N.

Exercice 13

Ecrire un programme `conversion.c` demandant une base `b`, comprise entre deux et trente-six, puis un nombre entier `n`, écrit en base dix, compris entre `2` et `b-1`, et affichant ce nombre dans la base `b`. Pour les trente-six bases, on utilisera les chiffres décimaux habituels et les lettres de l'alphabet, le caractère `A` désignant le chiffre dix dans la base onze, `B` le chiffre onze dans la base douze, etc, `Z` le chiffre trente-cinq dans la base trente-six.

Exercice 14

L'algorithme de `Babylone` calcule la racine carrée d'un nombre `A` avec une précision `P`. Il utilise une suite de nombres réels X_n tels que $X_{n+1} = (X_n + A/X_n)/2$. En C, programmer l'algorithme de `Babylone` en respectant les entrées sorties suivantes:

```
Calcul de la racine carree d'un nombre A avec une precision P.
Nombre A ? 2
Precision P ? 0.000001
Valeur initiale ? 1.8
x1 = 1.4555555582
erreur = 0.3444443941
x2 = 1.4148006439
erreur = 0.0407549143
x3 = 1.4142136574
erreur = 0.0005869865
```

L'utilisateur entre le nombre `A`, la précision `P` et la valeur initiale X_0 de la suite. A chaque itération, le programme affiche la valeur de X_n et l'erreur `e` $= |X_n - X_{n-1}|$ avec 10 décimales. Le programme s'arrête lorsque l'erreur `e` est inférieure à `P`. On pourra utiliser des variables `a`, `p`, `x`, `xsave`, `e` et `n`. On n'utilisera pas de fonction, ni de tableau.

Exercice 15

Ecrire un programme affichant toutes les solutions de l'équation `(1):  A x B = C`
où `A`, `B` et `C` sont trois chiffres distincts compris entre `1` et `9`. Le programme utilisera des boucles `for` et donnera la sortie suivante: `2x3=6. 2x4=8. 3x2=6. 4x2=8.`