Programmation 2

Feuille de TD-TP n°2

Récursivité terminale et non terminale

Exercice 1 (TD). Calcul de la n^e puissance d'un réel

Dans cet exercice, x désigne un nombre réel et n un nombre entier naturel. Si $n = 0$, on sait que $x^n = 1$.

a) Si $n > 0$, rappelez la relation de récurrence entre x^n et x^{n-1} puis écrivez la définition d'une fonction **récursive** `puiss` qui reçoit en arguments une valeur de type `double` x et une valeur de type `unsigned` n et renvoie la valeur de x^n.

Écrivez un programme qui initialise une variable x à 2 et une variable n à 6 puis affiche la valeur de x^n en appelant la fonction `puiss`. Simulez l'exécution de ce programme en représentant les états successifs de la pile d'appel.

Pour un entier naturel n arbitraire, évaluez, en fonction de n, le nombre d'appels récursifs à `puiss` exécutés si l'appel initial est `puiss(x, n)`.

b) Si $n > 0$, écrivez la relation de récurrence entre x^n et $x^{\lfloor n/2 \rfloor}$ puis écrivez la définition d'une fonction **récursive** `puiss_rap` qui reçoit en arguments une valeur de type `double` x et une valeur de type `unsigned` n et renvoie la valeur de x^n.

Écrivez un programme qui initialise une variable x à 2 et une variable n à 6 puis affiche la valeur de x^n en appelant la fonction `puiss_rap`. Simulez l'exécution de ce programme en représentant les états successifs de la pile d'appel.

Pour un entier naturel n arbitraire, évaluez, en fonction de n, le nombre d'appels récursifs à `puiss_rap` exécutés si l'appel initial est `puiss_rap(x, n)`. [Indication. Représentez n en base deux puis simulez les appels successifs. On rappelle que le nombre de bits nécessaire et suffisant pour représenter un entier $n > 0$ est $\lfloor \log_2 n \rfloor + 1$.]

c) Soit la fonction q définie par la relation de récurrence

$$q(x, n, y) = \begin{cases} y & \text{si } n = 0, \\ g(x, n-1, x * y) & \text{si } n \geqslant 1. \end{cases}$$

N.B. Le premier et le dernier argument de q sont des réels et le deuxième est un entier naturel. La récurrence ne porte que sur le deuxième argument.

Que vaut $q(2, 5, 1)$? En général, que vaut $q(x, n, 1)$ si $n > 0$? Écrivez la définition d'une fonction `puiss_ter` qui reçoit en arguments une valeur de type `double` x et une valeur de type `unsigned` n et renvoie la valeur de x^n. Cette fonction se contente d'appeler une fonction auxiliaire **récursive terminale** et de renvoyer le résultat de cette dernière.

d) La fonction `puiss_ter` de la question (*c*) implémente à l'aide d'une fonction auxiliaire récursive terminale la récurrence de la question (*a*). En appliquant la même méthode, écrivez la définition d'une fonction `puiss_rap_ter` qui implémente à l'aide d'une fonction auxiliaire récursive terminale la récurrence de la question (*b*).

Exercice 2 (TP). Récursivité terminale et compilation

Téléchargez depuis Moodle le programme `betise.c`. Comme son nom le laisse entendre, c'est un programme « bête » (un exemple à ne pas reproduire) qui empile indéfiniment les appels récursifs. Compilez-le en lançant `gcc` avec le « drapeau » (*i.e.* l'option) `-O0` (lettre O majuscule suivie du chiffre 0, correspondant au niveau d'optimisation le plus bas) puis exécutez-le. Que se passe-t-il ? Recompilez le programme source en lançant `gcc` avec l'option `-O2` puis relancez l'exécution. Que constatez-vous ? Donnez une explication simple mais percutante. [Indication. Le message « Erreur de segmentation » avertit que le processus a tenté d'accéder à une zone mémoire qui ne lui appartient pas.]

Commandes du shell. Vous pouvez arrêter l'exécution d'une commande en entrant au clavier `Ctr-C` et afficher la valeur renvoyée par la dernière commande exécutée en lançant `echo $?`

Remarque. Si vous utilisez un compilateur C différent du Gcc GNU des salles machines de l'institut Galilée, il se peut que, quand vous compilez avec l'option `-O2`, le compilateur « reconnaisse » que la fonction `betise` empile les appels récursifs sans effectuer d'autre action et traduise le programme source en un exécutable qui se contente de renvoyer 0. C'est le cas par exemple si vous avez un MacBook et que la version de Gcc est celle qui est livrée avec Xcode [1]. Dans ce cas, téléchargez depuis Moodle le programme `betise_bis.c` (qui n'est pas moins « bête » que l'autre) et recommencez toutes les opérations décrites ci-dessus.

Exercice 3 (TP). Calcul de la n^e puissance d'un réel

Écrivez un programme `puissance.c` qui lit un réel x et un entier positif n entrés au clavier par l'utilisateur, puis calcule et affiche la valeur de x^n en appelant successivement les quatre fonctions de l'exercice 1 de la feuille (traité en TD). Compilez-le puis testez l'exécutable.

Remarque. Pour connaître le nombre d'appels à chaque fonction pendant l'exécution de votre programme, vous pouvez utiliser des « compteurs », sous la forme de variables globales initialisées à 0 et incrémentées à chaque appel.

Vous pouvez aussi utiliser l'outil de profilage de Gcc, `gprof`. À cette fin, vous devez compiler votre programme source avec l'option `-pg`

```
gcc -Wall -ansi -Wfatal-errors -pg puissance.c -o prog_puissance
```

(`prog_puissance` est donc le nom de l'exécutable engendré si la compilation a réussi)
puis lancer l'exécutable

```
./prog_puissance
```

Vérifiez alors que le fichier `gmon.out` apparaît dans votre répertoire de travail. Lancez la commande

```
gprof -b prog_puissance gmon.out > analyse.out
```

(l'option `-b` permet d'éliminer des passages particulièrement « verbeux » de l'analyse)
et affichez, par exemple en lançant la commande `less analyse.out`, le fichier d'analyse.

1. À vrai dire, il ne s'agit pas d'une version de Gcc mais d'un autre compilateur, Clang, développé au sein d'un projet distinct de Gcc.