Travaux Dirigés d'algorithmique n°2

Cours d'informatique de Deuxième Année

—DEUG Sciences S3-MI—

Récursivité

La récursivité est une méthode de programmation très puissante. Elle permet d'écrire des programmes faciles à comprendre et souvent efficaces bien que son principal inconvénient est d'obliger le compilateur à utiliser une pile pour mémoriser les calculs intermédiaires.

▸ Exercice 1. Factorielle

La fonction factorielle *d'un entier positif ou nul n, notée $n!$, est définie par*

$$n! = \begin{cases} 1 & \text{si } n = 0, \\ (n-1)! * n & \text{si } n > 0. \end{cases}$$

Ecrire une fonction C, récursive, qui calcule la factorielle d'un entier positif. Ecrire une fonction C, itérative, qui calcule la factorielle d'un entier.

▸ Exercice 2. Puissance

Soient a un entier et n un entier positif ou nul. On se propose de calculer a^n en utilisant les trois propriétés suivantes :

1. $a^n = \begin{cases} a * a^{n-1} & \text{si } n > 0, \\ 1 & \text{si } n = 0. \end{cases}$
2. $a^n = \begin{cases} (a^2)^{\lfloor n/2 \rfloor} & \text{si } n > 0 \text{ est pair}, \\ a * (a^2)^{\lfloor n/2 \rfloor} & \text{si } n \text{ est impair}, \\ 1 & \text{si } n = 0. \end{cases}$
3. $a^n = \begin{cases} (a^{\lfloor n/2 \rfloor})^2 & \text{si } n > 0 \text{ est pair}, \\ a * (a^{\lfloor n/2 \rfloor})^2 & \text{si } n \text{ est impair}, \\ 1 & \text{si } n = 0. \end{cases}$

écrire, pour chacune des trois propriétés susnommées, une fonction C récursive qui retourne a^n en utilisant cette propriété.

▸ Exercice 3. Etrange

Après exécution du programme suivant, qu'a-t-on en sortie ? Que fait la fonction `Etrange` *et en particulier, à quel moment a lieu le "passage à la ligne" ?*

```
#include <stdio.h>

void Etrange(int n)
{
  if (n <= 0)
    printf("\n");
  else
    { Etrange(n-1); printf("%d ", n);}
}
```

En déduire une fonction récursive qui affiche les n premiers entiers dans l'ordre croissant et une autre qui les affiche dans l'ordre décroissant.

▸ Exercice 4. Recherche dichotomique

On rappelle que l'idée de la recherche par dichotomie d'un élément x dans un tableau trié pour l'ordre croissant est la suivante :

on compare x avec l'élément m du milieu du tableau,

- *si $x = m$: on a trouvé une solution ;*
- *si $x < m$: x ne peut pas se trouver après m dans le tableau; il suffit alors de le rechercher dans la partie à gauche de m ;*
- *si $x > m$: x ne peut pas se trouver avant m dans le tableau; il suffit alors de le rechercher dans la partie à droite de m ;*

Quand peut-on conclure que x n'est pas présent dans le tableau ?

Donner une fonction récursive qui traduit cet algorithme.

▸ Exercice 5. Monnayeur

Problème: étant donnée une suite d'entiers strictement croissante $P_0 = 1, P_1, \ldots, P_{N-1}$ et un entier strictement positif S, déterminer un ensemble d'entiers positifs $\{a_0, a_1, \ldots, a_{N-1}\}$ tels que $\sum_{i=0}^{N-1} a_i P_i = S$ avec $\sum_{i=0}^{N-1} a_i$ minimale. Il y a toujours une solution puisque $P_0 = 1$.
On dispose d'un tableau `piece[N]` *contenant les entiers P_i en ordre croissant.*

1. *Écrire une fonction* `int Glouton(int s, int valeur[])` *qui réalise la somme* `s` *avec les entiers du tableau* `valeur`*, sans se préoccuper de la condition de minimalité, en commençant systématiquement par utiliser les plus grandes valeurs.* *L'exemple* `piece=` $\boxed{1 \quad 5 \quad 6}$ *et $S = 10$ permet de se convaincre que la méthode n'est pas optimale.*
2. *Écrire une fonction* `int Monnaie(int s, int valeur[],int indiceMax)` *réalisant le minimum. Pour cela elle compare le nombres de pièces obtenu en utilisant les valeurs jusqu'à l'indiceMax avec le nombre de pièces obtenus en utilisant les valeurs jusqu'à l'indiceMax -1.*