ECE Lyon — ING1 — Informatique 2 — Devoir Surveillé n°4
Récursivité et structures de données (listes chaînées)

Partie 1 - Récursivité *(12 points)*

Exercice 1 - Cours *(2 points)*

— Comparez les implémentations d'une même fonction en version itérative et en version récursive.
— Donnez l'inconvénient principal de l'implémentation récursive.

Exercice 2 - Récursivité directe *(4 points)*

La fonction d'*Ackermann* est un exemple simple de fonction récursive directe non récursive primitive, c'est-à-dire une fonction construite à partir de la fonction nulle, de la fonction successeur, des fonctions projections et des schémas primitifs de récursion et de composition.

Cette fonction est définie pour $n, m \in \mathbb{N}$ par :

$$Ackermann(m,n) = \begin{cases} n+1 & \text{, si } m = 0 \\ Ackermann(m-1,1) & \text{, si } m > 0 \text{ et } n = 0 \\ Ackermann(m-1, Ackermann(m,n-1)) & \text{, si } m > 0 \text{ et } n > 0 \end{cases}$$

Exercice 2.1 - Implémentation d'*Ackermann* *(1,5 point)*

Donnez le code d'un sous-programme qui implémente cette fonction.

Exercice 2.2 - Trace mémoire *(1,5 point)*

Donnez la trace mémoire et le résultat d'$Ackermann(1,1)$. Vous utiliserez le tableau à 5 colonnes (`Adresse`, `Valeur`, `Variable`, `Fonction`, `Return`) fourni en annexe.

Exercice 2.3 - Empilement/dépilement *(1 point)*

Donnez l'empilement et le dépilement d'$Ackermann(1,1)$.

Exercice 3 - Récursivité indirecte *(6 points)*

La vérification de la parité d'un nombre peut être effectuée au moyen de fonctions récursives indirectes.

Ces fonctions sont définies pour $n, m \in \mathbb{N}$ par :

$$Pair(n) = \begin{cases} Vrai & \text{, si } n = 0 \\ Faux & \text{, si } n = 1 \\ Impair(n-1) & \text{, si } n > 1 \end{cases} \qquad Impair(m) = \begin{cases} Vrai & \text{, si } m = 1 \\ Faux & \text{, si } m = 0 \\ Pair(m-1) & \text{, si } m > 1 \end{cases}$$

3.1) Implémentation de *Pair* et *Impair* *(1,5 point)*

Donnez le code de deux sous-programmes implémentant ces deux fonctions.

3.2) Trace mémoire *(1,5 point)*

Donnez la trace mémoire de $Pair(3)$. Vous utiliserez le tableau à 5 colonnes (`Adresse`, `Valeur`, `Variable`, `Fonction`, `Return`) fourni en annexe.

3.3) Empilement/dépilement *(1 point)*

Donnez l'empilement et le dépilement de $Pair(3)$.

3.4) Parité par itération *(2 points)*

Donnez le code d'un sous-programme permettant de vérifier la parité d'un nombre de manière itérative, sans utiliser l'opérateur modulo (`%`).

Partie 2 - Listes chaînées *(8 points)*

Exercice 4 - Structures de données *(2 points)*

— Comparez l'utilisation de tableaux dynamiques avec l'utilisation de listes chaînées. Vous donnerez les avantages et inconvénients de chacun.
— Quand utilise-t-on plutôt l'une de ces deux structures de données plutôt que l'autre ?

Exercice 5 - Insertion en milieu de liste simplement chaînée *(2 points)*

Donnez les schémas successifs des étapes permettant d'insérer un élément (cellule/maillon) entre la 2ème et la 3ème cellule d'une liste simplement chaînée (initialement composée de 3 cellules). Votre algorithme devra fonctionner quel que soit le nombre de cellules présentes initialement dans la liste chaînée (0, 1, ou plusieurs). Vous accompagnerez vos schémas de textes lorsque cela est nécessaire à la compréhension de l'algorithme.

Exercice 6 - Insertion en début de liste simplement chaînée *(2 points)*

Donnez le code d'un sous-programme permettant d'insérer un élément (cellule/maillon) au début d'une liste simplement chaînées. Ce sous-programme devra fonctionner quel que soit le nombre d'éléments présents initialement dans la liste chaînée (0, 1, ou plusieurs).

Exercice 7 - Définitions de liste doublement chaînée *(2 points)*

Donnez le code permettant de définir un type `Cellule`, représentant chaque maillon d'une liste doublement chaînée. Chaque `Cellule` devra contenir un `nom` (chaîne d'au plus 100 caractères) et un `niveau` (entier positif). Vous donnerez également le code du programme principal déclarant une liste doublement chaînées de `Cellules`, **vide** (initialisée sans aucune `Cellule`).