TD3 : Entiers arbitrairement grands

11 mars 2011

Les entiers directement utilisables en C sont codés sur un nombre borné de bits (entre 8 et 128 au maximum selon la machine). Les entiers représentables sont donc compris dans un intervalle prédéfini. Nous proposons dans ce TD d'étudier un codage des entiers utilisant les listes vues au TD précédent pour représenter des entiers de taille arbitraire.

Soit $x \in \mathbb{N}$, et soit $a_k a_{k-1} \dots a_1 a_0$ sa représentation décimale. On a bien sûr $0 \leq a_i \leq 9$ pour tout i et $k = \lfloor \log_{10} x \rfloor + 1$. Le codage proposé consiste simplement à construire la liste $a_0 :: a_1 :: \dots :: a_{k-1} :: a_k :: \emptyset$. Par exemple, le nombre 546 sera représenté par la liste $6 :: 4 :: 5 :: \emptyset$.

1. Écrivez une fonction `list of_int(int x)` qui construit la liste correspondant à l'entier (positif) `x`.
2. Écrivez une fonction `void print(list l)` qui affiche à l'écran l'entier codé par la liste `l`.
3. Écrivez une fonction `int to_int(list l)` qui calcule l'entier correspondant à la liste donnée en entrée. Grâce aux questions précédentes, proposez une fonction de test convaincante pour votre implémentation.
4. [Difficile] On se propose maintenant d'implémenter l'addition de deux listes. La fonction d'addition devra créer une nouvelle liste pour fournir son résultat. La difficulté majeure est de faire circuler la retenue correctement, même lorsque l'on atteint la fin de l'une ou des deux listes.
5. De même que précédemment, proposer un test convaincant de votre implémentation.
6. [Optionnel et difficile] Proposez une implémentation alternative de l'addition où le premier argument est modifié pour rendre le résultat, c'est-à-dire que la fonction `addition2(x,y)` modifie la liste `x` de sorte qu'elle représente la somme des deux entiers correspondant aux listes. Veillez bien à ne pas créer de fuite mémoire ! Quel avantage cette implémentation présente-t-elle ?
7. En vous servant de l'addition précédemment implémentée, écrivez la multiplication, puis la factorielle.
8. Écrivez un programme de test mettant en évidence la correction de votre code, ainsi que la différence de performance entre les calculs sur listes et les calculs sur les types primitifs du C.