Rapport de TP

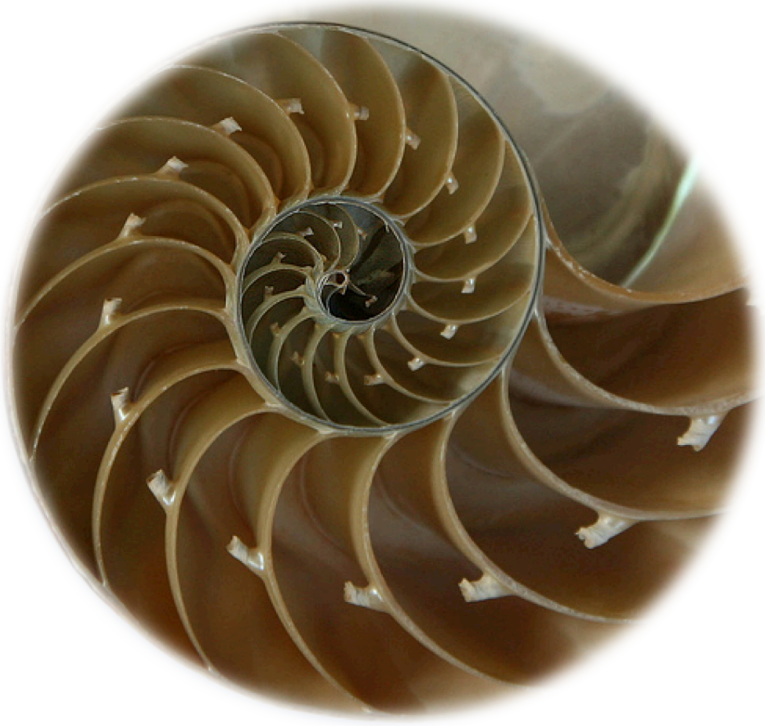
Arbres et Fibonacci

Oumaima Talouka – GI01

Guillaume Olympe – GB04

Daniel Suárez Escudero – GB04

Encadrante : Mme. El Hajj



P15

utc
Université de Technologie
Compiègne

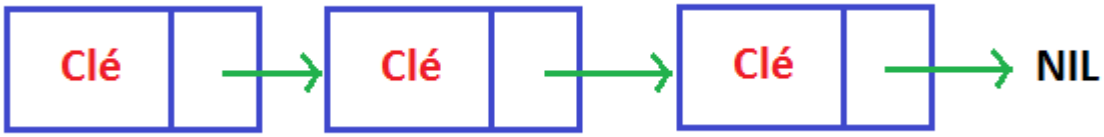
Introduction

Le but de ce TP est de programmer un **arbre binaire** dont le nombre de symboles présents sur chaque niveau k représente le nombre correspondant à la suite de **Fibonacci** au niveau k. La génération de l'arbre suit le système de Lindenmayer. Bien sûr, l'arbre peut contenir un nombre indéfini de niveaux à déterminer par l'utilisateur.

Chaque nœud de l'arbre contient une lettre (soit A, soit B), et pointe vers son fils gauche, vers son fils droit et vers son père. Un nœud A aura un fils gauche B et pas de fils droit, et un nœud B aura un fils gauche A et un fils droit B. Il est à noter qu'à l'initialisation l'arbre contient un seul nœud A.

Il est affaire également de créer des listes permettant d'organiser les éléments de chaque niveau de l'arbre, ceci en se servant notamment de listes simplement chaînées. De cette manière, chaque nœud pointe également vers le nœud à sa droite (sur le même niveau).

Pour rappel, les listes simplement chaînées sont une structure qui permet d'organiser facilement une suite d'éléments. Elles sont composées de cases, et chacune des cases contient une clé et un pointeur vers la case suivante. La dernière case pointe vers NIL. Dans le cadre de ce TP, nous connaissons les adresses de début et de fin de chaque liste.



Il sera ainsi affaire de créer une structure qui représentera un nœud de l'arbre, et une structure qui représentera une liste chaînée pour organiser une liste des symboles de chaque niveau k de l'arbre.

A noter que tout est codé en C.

Forme du programme

Le programme se compose de diverses fonctions qui permettent de créer des nœuds et des listes, d'ajouter des éléments dans les listes (ceci en respectant l'ordre imposé par le système Lindenmayer), de supprimer des éléments, d'afficher les éléments de l'arbre ou d'une liste et de calculer la taille d'une liste.

Le programme propose dans un menu principal les choix suivants :

1- Initialiser un arbre
2- Générer un arbre de niveau n, saisi par l'utilisateur
3- Ajouter un niveau
4- Supprimer le dernier niveau
5- Afficher la liste des feuilles de l'arbre
6- Afficher l'arbre
7- Calculer le Fibo d'un nombre n, saisi par l'utilisateur
8- Détruire l'arbre
9- Quitter

Fonctions

Liste des fonctions demandées :

createNode : Cette fonction permet de créer un nœud, ayant la lettre correspondante pour paramètre.

generatechilds : Cette fonction permet de créer les fils d'un nœud, en respectant l'ordre imposé par le système Lindenmayer. Les liens nécessaires sont créés.

freeTree : Cette fonction libère un arbre créé.

displayTree : Cette fonction affiche un arbre de manière à représenter chaque noeud n, par la notation {g,n,d} où g est le sous-arbre gauche, n le contenu du noeud et d le sous-arbre droit.

createList : Cette fonction permet de créer une liste vide.

addEndList: Cette fonction permet d'ajouter un nœud à la fin d'une liste.

generateList : En se plaçant sur un niveau k, cette fonction permet de générer la liste des feuilles du niveau k+1.

removeList : Cette fonction permet de supprimer une liste. Elle « coupe » les liens de chaque nœud à éliminer dans la liste chaînée et libère la mémoire qu'il utilise.

displayList : Cette fonction affiche les éléments contenus dans une liste.

calculateSizeList : cette fonction permet de calculer la taille d'une liste.

Fonction bonus :

Inverse : Cette fonction crée un nouvel arbre à partir d'un arbre initial, et les lettres sont inversées (les A deviennent des B et vice-versa). Elle affiche ensuite l'ancien et le nouvel arbre. Pour ne pas avoir des problèmes en mémoire, nous avons décidé de créer un nouvel arbre, identique à l'arbre initial, et d'inverser ce deuxième. Il aurait été également possible d'utiliser une structure de piles, on empile en inversant les sommets, puis on dépile chaque sommet et nous l'insérons dans l'arbre.

Liste des fonctions supplémentaires :

DisplayTreeInv : Cette fonction nous permet d'afficher l'arbre inversé suite à la fonction *Inverse*

Complexité

Fonctions demandées :

createNode : O(1).

generatechilds: O(1).

freeTree : O(n) où n est le nombre de nœuds dans la liste de niveau k.

displayTree : O(n) où n est le nombre de nœuds dans la liste de niveau k.

createList : O(1).

addEndList : O(1).

generateList : O(n) où n est le nombre de nœuds dans la liste de niveau k.

removeList : O(n) où n est le nombre de nœuds dans la liste à supprimer.

displayList : O(n) où n est le nombre de nœuds dans la liste. Cependant, bien qu'il y ait n itérations, on pourrait considérer une complexité O(1) du fait qu'il n'y a pas d'affectations dans la fonction.

calculateSizeList : O(n) où n est le nombre de nœuds dans la liste.

Bonus :

Inverse : O(n) où n est le nombre de nœuds dans la liste.

Fonction Supplémentaire :

DisplayTreeInv : O(n) où n est le nombre de nœuds dans la liste de niveau k.

Conclusion

Ce TP nous a permis d'appréhender la manipulation des arbres binaires en C, ainsi que de renforcer nos connaissances sur les listes chaînées. Nous avons notamment manipulé des pointeurs, des fonctions et des structures, en les mettant en commun pour créer un programme efficace et fonctionnel. Enfin, nous avons appris à réfléchir en matière de d'amélioration de nos algorithmes dans le but de réduire la complexité des fonctions codées.