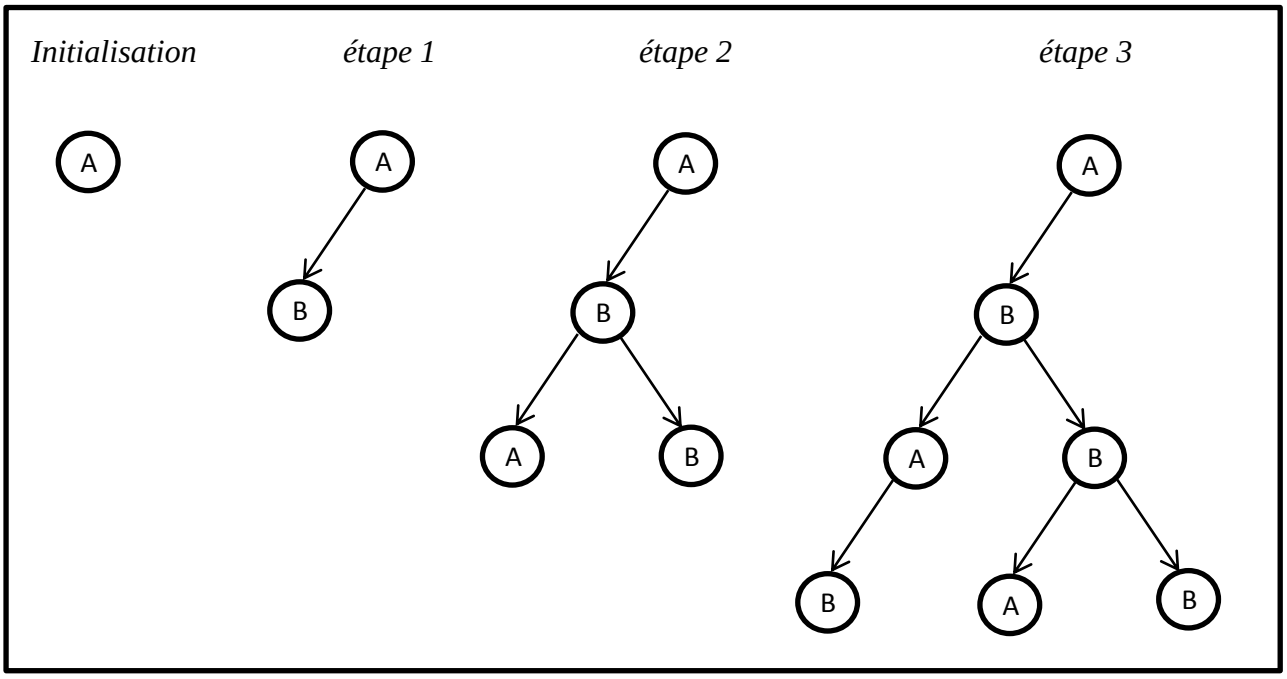
Problème arbre binaire
Suite de Fibonacci

Les éléments de la suite de Fibonacci peuvent être déterminés en utilisant la technique du L-system. Par définition, un L-System ou système de Lindenmayer est une grammaire formelle, inventée par le biologiste hongrois Aristid Lindenmayer, qui consiste à modéliser le processus de développement et de prolifération de plantes ou de bactéries. L'arbre binaire construit en se basant sur le L-system a les caractéristiques suivantes :

- Chaque nœud contient une lettre, soit A ou B, et peut avoir au maximum 2 fils, noté fils gauche et fils droit.
- A l'initialisation, l'arbre contient un seul nœud avec la lettre A. Ensuite la construction de l'arbre se fait étape par étape en générant à chaque fois les fils des feuilles de l'étape précédente.
- La règle de génération est la suivante : une feuille A va générer un fils gauche B et pas de fils droit, une feuille B va générer un fils gauche A et un fils droit B.

En utilisant le L-System, nous générons un arbre binaire dont le nombre de symboles présents sur chaque niveau ***i*** représente le nombre correspondant à ***Fibo(i)***.

La figure suivante représente l'arbre généré jusqu'au troisième niveau, suivie par le résultat obtenu pour les 6 premières étapes.



Voici le résultat sur six générations :

- niveau 1, A ⟶ 1 symbole, $Fibo(1) = 1$
- niveau 2, B ⟶ 1 symbole, $Fibo(2) = 1$
- niveau 3, AB ⟶ 2 symboles, $Fibo(3) = 2$
- niveau 4, B AB ⟶ 3 symboles, $Fibo(4) = 3$
- niveau 5, AB B AB ⟶ 5 symboles, $Fibo(5) = 5$
- niveau 6, B AB AB B AB ⟶ 8 symboles, $Fibo(6) = 8$

Pour réduire la complexité de la création d'un niveau supplémentaire, nous utilisons une liste qui permet de chaîner tous les nœuds d'un niveau donné de l'arbre. Dans la suite, on considère les structures de données suivantes :

```
typedef struct Node {
        char letter;
        // pour l'arbre
        struct Node* parent;
        struct Node * left;
        struct Node * right;
        // pour la liste
        struct Node * next ;
} Node;

typedef struct ListNode {
        struct Node * start;
        int number ;
} ListNode;
```

Fonctions à implémenter :

1- *Node * createNode(char letter)* qui crée un nœud avec la lettre donnée *letter* (A ou B).
2- *void generateChilds(Node * node*) qui génère les fils de la feuille *node* selon la règle de L-system.
3- *void freeTree(Node * root)* qui libère l'arbre.
4- *void displayTree(Node * root)* qui affiche l'arbre de manière à représenter chaque noeud v par {g,v,d} où g est le sous-arbre gauche, v la valeur du nœud et d le sous-arbre droit. Les "-" indiquent les sous-arbres vides. Par exemple l'arbre de la figure 1 sera affiché de la manière suivante: {{{B,A,-},B,{A,B,B}},A,-}
5- *ListNode * createList()* pour créer une liste vide.

6- *void addEndList(ListNode * list, Node * node)* pour ajouter la feuille *node* à la fin de la liste *list*.
7- *ListNode * generateList(ListNode * list)* pour générer la prochaine liste des feuilles en se basant sur la liste actuelle. Cette fonction devrait utiliser la fonction *generateChilds(Node * node)*.
8- *void removeList(ListNode * list)* pour supprimer la liste des feuilles *list*.
9- *void displayList(ListNode * list)* pour afficher la liste des feuilles *list*.
10- *int calculateSizeList(ListNode * list)* pour calculer la taille d'une liste *list*.

Programme principal

Utiliser les fonctions développées pour écrire le programme *main* qui permet à l'utilisateur de générer et de manipuler l'arbre en lui offrant les possibilités suivantes :

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

Bonus

Écrire la fonction *Node * Inverse(Node * node)* qui inverse toutes les lettres de l'arbre (le A devient un B et le B devient un A). Afficher ensuite l'ancien et le nouvel arbre.

Dossiers à déposer :

L'organisation **MINIMALE** du projet sous Visual Studio ou CodeBlocks est la suivante :

- Fichier d'en-tête tp4.h, contenant la déclaration des structures/fonctions de base,
- Fichier source tp4.c, contenant la définition de chaque fonction,
- Fichier source main.c, contenant le programme principal.

Avec ces trois fichiers, un compte rendu, de 2 à 3 pages, doit être présenté, comportant principalement :

- les spécifications précises de chaque fonction : ses entrées, ses sorties et une description détaillée des différentes procédures et techniques utilisées (pas de code dans le compte rendu),
- le calcul des complexités des différentes procédures réalisées.