Sujet G

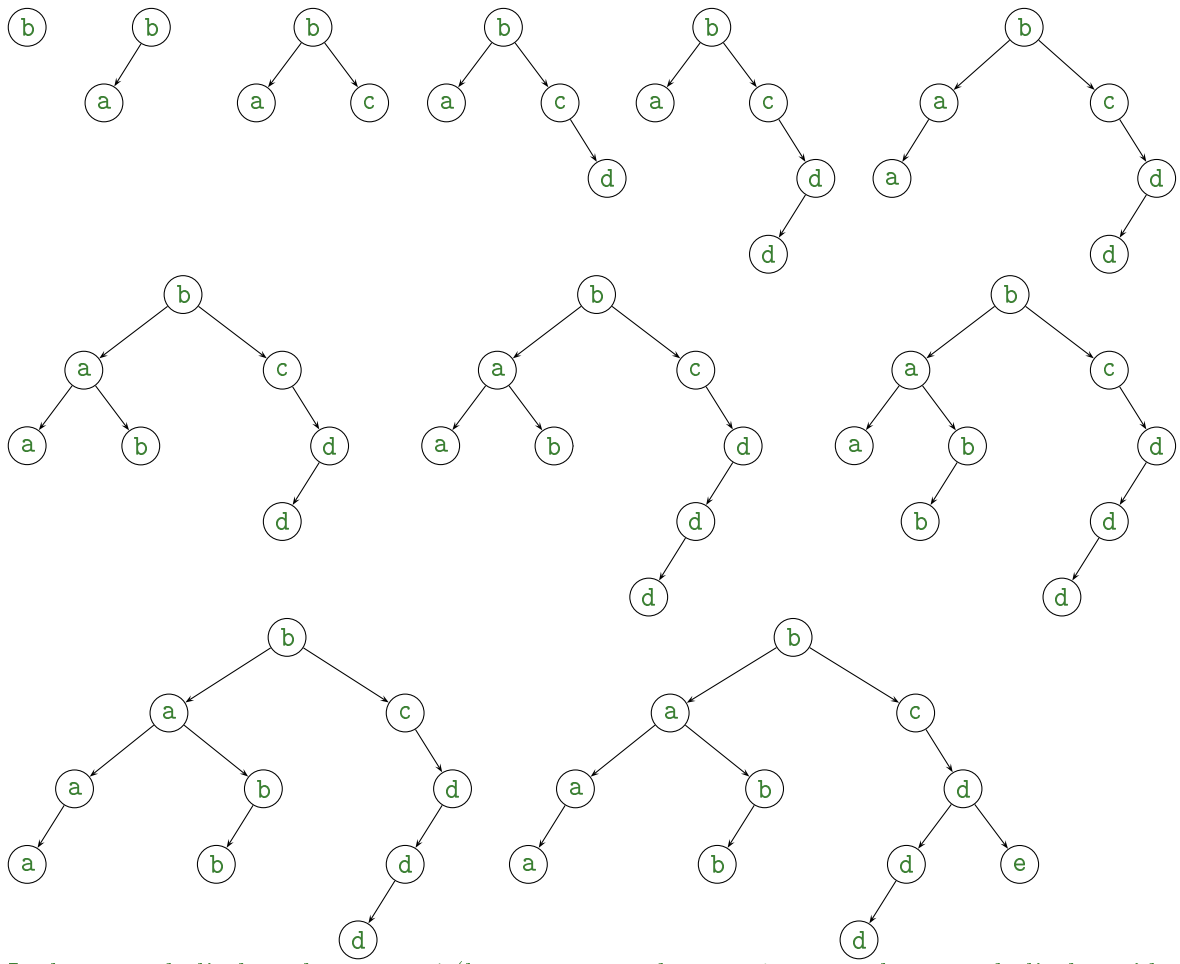
❑ **Exercice** : *type A*

Dans cet exercice, on autorise les doublons dans un arbre binaire de recherche et pour le cas d'égalité on choisira le sous-arbre gauche. On ne cherchera pas à équilibrer les arbres.

1. Rappeler la définition d'un arbre binaire de recherche.

   Un arbre binaire de recherche sur un ensemble d'étiquettes E (totalement ordonné), peut se définir inductivement par l'axiome $\varnothing$ (arbre vide), et la règle d'inférence d'arité 2 : $(g, d) \mapsto (g, x, d)$ où $x \in E$ et toute étiquette de g est inférieure à x et toute étiquette de d est supérieure à x.

2. Insérer successivement et une à une dans un arbre binaire de recherche initialement vide toutes les lettres du mot `bacddabdbae`, en utilisant l'ordre alphabétique sur les lettres. Quelle est la hauteur de l'arbre ainsi obtenu ?



   La hauteur de l'arbre obtenu est 4 (la programme donne -1 comme hauteur de l'arbre vide.)

3. Montrer que le parcours en profondeur infixe d'un arbre binaire de recherche de lettres est un mot dont les lettres sont rangées dans l'ordre croissant. On pourra procéder par induction structurelle.

On procède par induction structurelle, pour tout ABR a, on note $P(a)$ la propriété « le parcours infixe de a est un mot dont les lettres sont rangées dans l'ordre croissant ».

— $P(\varnothing)$ est vraie et donc la propriété P est vérifiée pour tous les axiomes.

— Montrons à présent la conservation de la propriété P par application de la règle d'inférence, si g et d sont deux ABR vérifiant P, et x une lettre telle que toutes les lettres de g sont avant x et toutes les lettres de d sont après x dans l'ordre alphabétique. Le parcours infixe de l'ABR (g, x, d) est par définition le parcours infixe de g suivi de x suivi du parcours infixe de d. Par hypothèse d'induction, les lettres du parcours de g et celles du parcours de d sont rangées par ordre croissant. Comme de plus x est après toute les lettres de g et avant toutes celles de d, le parcours de (g, x, d) est bien formé de lettres rangées par ordre croissant.

Par application du principe d'induction structurelle, pour tout ABR a, $P(a)$ est vraie.

4. Proposer un algorithme qui permet de compter le nombre d'occurrences d'une lettre dans un arbre binaire de recherche de lettres. Quelle est sa complexité ?

On part de la racine, à chaque étape, on ajoute 1 au nombre d'occurence si l'étiquette est la lettre cherchée et on descend dans le sous arbre gauche si l'étiquette est inférieure ou égale à la racine et dans le sous arbre droit sinon. On s'arrête en rencontrant une feuille. Comme à chaque étape on descend d'un niveau dans l'orbre la complexité est en $O(h)$ où h est la hauteur de l'arbre.

❑ **Exercice** : *type B*

Le langage utilisé dans cet exercice est le langage C.

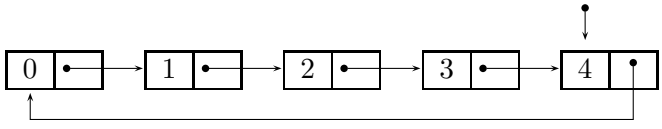
Certaines comptines enfantines ont pour objectif de désigner une personne « au hasard », un exemple bien connu est « *Am, stram, gram, pic et pic et colégram* ». On suppose que N enfants numérotés de 0 à $N-1$ sont assis en cercle et que l'un d'entre eux (le numéro k) récite une comptine contenant S syllabes. A la première syllable il désigne son suivant immédiat dans le cercle puis il avance d'un enfant à chaque syllable jusqu'à la fin de la comptine. L'enfant désigné à la fin de la comptine doit quitter le cercle et le processus recommence à partir de son suivant immédiat jusqu'à ce qu'un seul enfant reste.

1. Donner une illustration de ce processus avec $N = 5$ et $S = 7$, en supposant que l'enfant 0 commence et indiquer le numéro du dernier enfant restant.

- [0 ; 1 ; 2 ; 3 ; 4] 2 quitte le cercle et 3 récite la comptine
- [0 ; 1 ; 3 ; 4] 1 quitte le cercle et 3 récite la comptine
- [0 ; 3 ; 4] 4 quitte le cercle et 0 récite la comptine
- [0 ; 4] 4 quitte le cercle

Donc c'est l'enfant 0 qui reste.

On veut implémenter une structure de données de liste chainée circulaire permettant de représenter les enfants assis en cercle. On rappelle qu'une liste chainée circulaire est une liste dont le dernier maillon pointe vers le premier maillon. L'accès au cercle se fait via un pointeur qui désigne l'enfant qui *précède celui qui doit dire la comptine*. Par exemple un cercle initial de cinq enfants où le premier à dire la comptine est le numéro 0 est représenté par la structure de données ci-dessous :



Afin d'implémenter cette structure de données, on propose d'utiliser les types suivants

```c
struct maillon_s
{
    int valeur;
    struct maillon_s *suivant;
};
typedef struct maillon_s maillon;
typedef maillon *liste_circulaire;
```

Ce type est défini dans le fichier compagnon de cet exercice téléchargeable à l'adresse : `https://fabricenativel.github.io/cpge-info/oraux/`.
Ce fichier contient également une fonction de prototype `void affiche(liste_circulaire lc)` qui affiche le contenu de la liste circulaire à partir du maillon *qui suit* celui désigné par `lc`.

2. Ecrire une fonction de signature `void ajouter(liste_circulaire *lc, int v)` qui prend en paramètre un pointeur vers une liste circulaire et ajoute un maillon de valeur `v` après le maillon pointé par `lc` et met à jour le pointeur `lc` pour qu'il pointe vers le nouveau maillon.

```c
void ajouter(liste_circulaire *lc, int v)
{
    maillon *nm = malloc(sizeof(maillon));
    nm->valeur = v;
    if (est_vide(*lc))
    {
        nm->suivant = nm;
    }
    else
    {
        nm->suivant = (*lc)->suivant;
        (*lc)->suivant = nm;
    }
    *lc = nm;
}
```

3. En déduite une fonction de signature `liste_circulaire creer_cercle(int n)` qui prend en paramètre un entier `n` et qui crée un cercle de `n` enfants numérotés de 0 à `n-1`.

```c
liste_circulaire creer_cercle(int n)
{ // initialise une liste circulaire avec n maillons contenant les valeurs de 0 à
↪  n-1
    liste_circulaire cercle = NULL;
    for (int i = 0; i < n; i++)
    {
        ajouter(&cercle, i);
    }
    return cercle;
}
```

4. Ecrire une fonction de signature `void avance(liste_circulaire *lc, int s)` qui modifie la liste circulaire `lc` en avançant de `s` maillons.

```c
void avance(liste_circulaire *f, int s)
{

    if (!est_vide(*f))
    {
        for (int i = 0; i < s; i++)
        {
            *f = (*f)->suivant;
        }
    }
}
```

5. Ecrire une fonction de signature `void enleve(liste_circulaire *lc)` qui modifie la liste circulaire `lc` en enlevant le maillon *qui suit* celui pointé par `lc`.

```c
int enleve(liste_circulaire *f)
{
    int res = ((*f)->suivant)->valeur;
    maillon *old = (*f)->suivant;
    if ((*f)->suivant == *f)
    {
        *f = NULL;
    }
    else
    {
        (*f)->suivant = ((*f)->suivant)->suivant;
    }
    free(old);
    return res;
}
```

6. En déduire une fonction de signature `int dernier_enfant(int n, int s)` qui renvoie le numéro du dernier enfant restant dans le cercle de `n` enfants après avoir récité une comptine de `s` syllabes en supposant que l'enfant 0 commence.